TECHNICAL REPORT

ISO/IEC TR 11590

First edition
1995-10-01

Information technology — Open Systems Interconnection — LOTOS description of the CCR protocol

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — Description LOTOS du protocole CCR

Reference number
ISO/IEC TR 11590:1995(E)

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The main task of technical committees is to prepare International Standards. In exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 11590, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open systems interconnection, data management and open distributed processing*.

Introduction

This Technical Report describes ISO/IEC 9805-1 CCR Protocol Specification using LOTOS.

LOTOS is a formal description technique(FDT) to define behavior of systems with a formal syntax and semantics.

Aim of this Technical Report is to give an unambiguous and precise description of ISO/IEC 9805-1.

However, in case of inconsistency between ISO/IEC 9805-1 and this Technical Report, ISO/IEC 9805-1 takes precedence over this Technical Report.

This Technical Report consists of as follows:

- In clauses 5, 6, and 7, the description that is, the structures of processes, types, and gates, are explained.
- In clause 8, the types which are used in the description are defined.
- In clause 11, global constraints of the CCR protocol specification are described.
- In clause 12, a CCR protocol machine is described.
- In clause 13, CF (control function) is described.

Information technology — Open Systems Interconnection — LOTOS description of the CCR protocol

1 Scope

This Technical Report describes ISO/IEC 9805-1 CCR Protocol Specification using formal description language LOTOS. The LOTOS specification includes not only CCR protocol machine, but also CF(control function) which controls the ASEs in order that the architecture of the CCR-ASE process is compatible with the ALS structure.

According to the ALS structure, this Technical Report employs the model as shown in Clause 6. LOTOS Specification covers CF(control function) as well as CCR Protocol machine, however, the description of CF includes only its functions concerning on CCR operations.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9804:1994, *Information technology – Open Systems Interconnection – Service definition for the Commitment, Concurrency and Recovery service element.*

ISO/IEC 9805-1:1994, *Information technology – Open Systems Interconnection – Protocol for the Commitment, Concurrency and Recovery service element: Protocol Specification.*

ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.*

3 Definitions

For the purposes of this Technical Report the definitions given in ISO/IEC 9805-1 apply.

4 Symbols and abbreviations

The abbreviations given in ISO/IEC 9805-1 apply.

The abbreviation "LOTOS" is defined in ISO 8807.

5 Conventions

Clause 7 through 13 of this Technical Report consists of a LOTOS text. All explanatory descriptions are caught between "(*" and "*)" symbols and are treated as comments of the LOTOS text.

6 Introduction to the formal description

6.1 Model

CCR protocol in this Technical Report is modeled as shown in Figure 1.

This Technical Report specifies the specification `CCR_Protocol_in_ALS` which has five gates, `u`, `acse_u`, `acse_l`, `u_env`, and `env`, and one parameter `ver`.

`CCR_Protocol_in_ALS` has two processes, `CCR` and `CF`. Process `CCR` is actually a protocol machine for the CCR operation. `CF` is a process to perform a control functions to be compatible with the ALS structure.

`CF` includes three processes, `CFRouter`, `ConcDeconcProc`, and `LowerMapper`. `CFRouter` controls process `CCR` via two gates, `ccr_u` and `ccr_l`. `CFRouter` should also support the behavior for ACSE or other parts, but those parts are not essential for the CCR specification. Therefore, they are omitted in this specification. Instead, ACSE can communicate via two gates, `acse_u` and `acse_l`.

`ConcDeconcProc` performs to concatenate and deconcatenate the PDUs for CCR and other ASEs and send it to the presentation layer via process `LowerMapper`.

Process `LowerMapper` maps in between the PDUs concatenated by `ConcDeconcProc` and the presentation services.

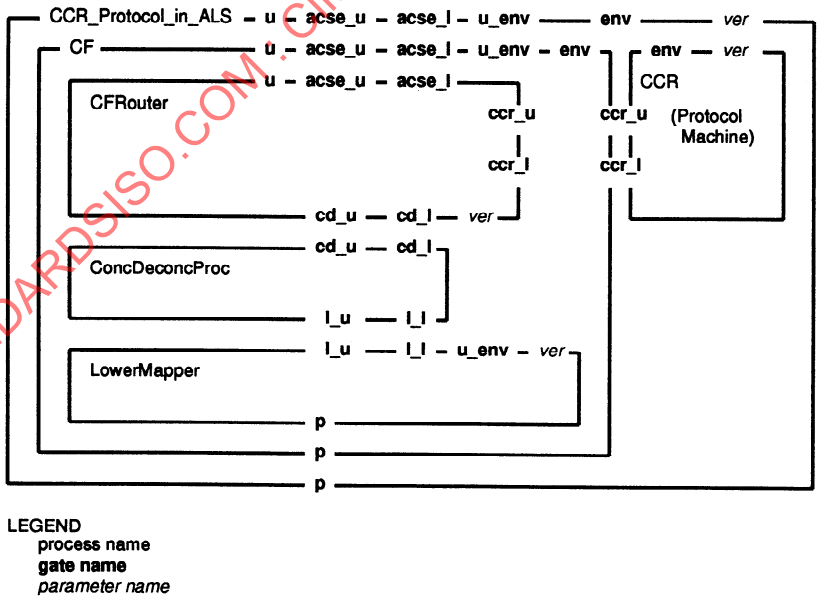

Figure 1 - CCR protocol model

6.2 Structure

The relationships among the processes and the types are shown in Figure 2 and 3, respectively. In these figures, "*" means that "the definition has already been listed above".

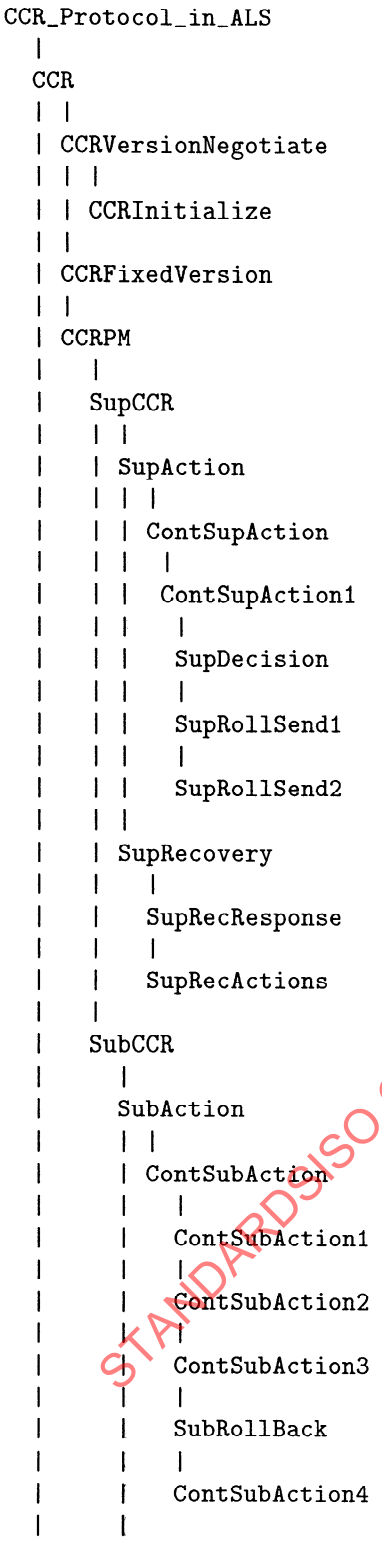

Figure 2 - Relationship among the processes

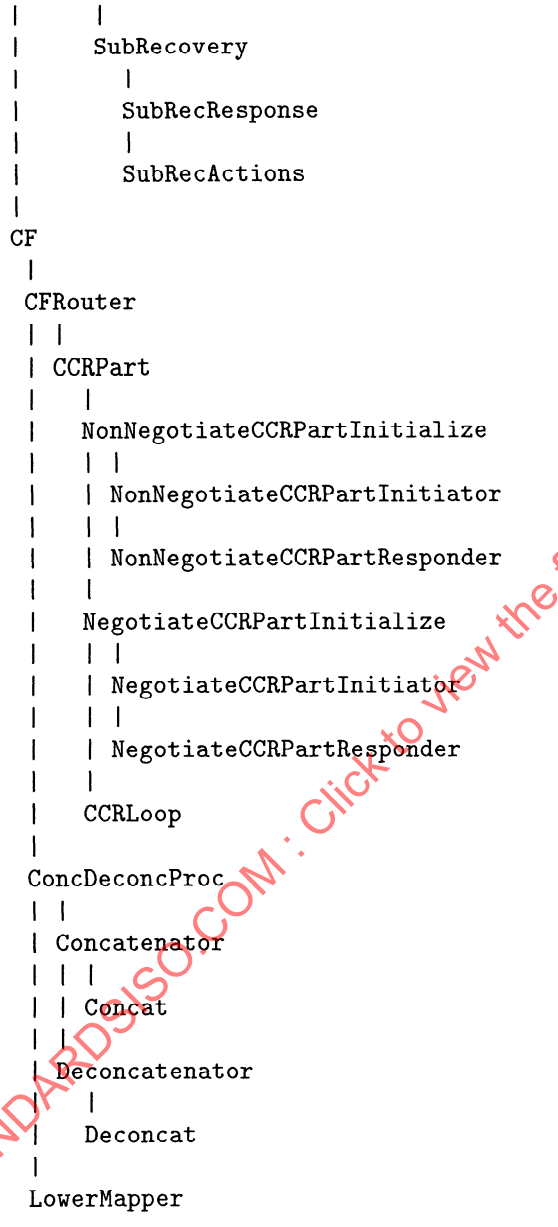

Figure 2 - Relationship among the processes (concluded)

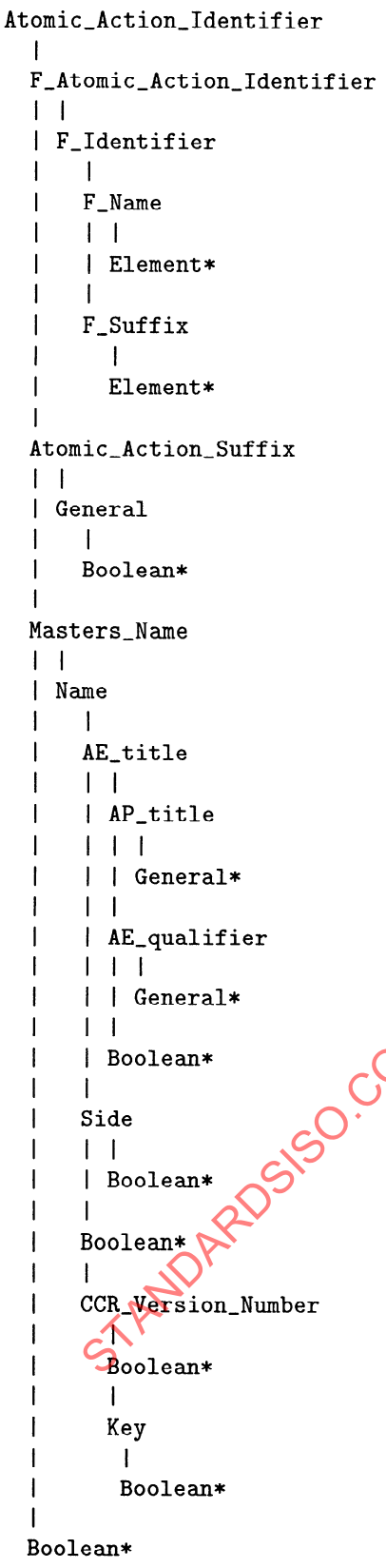

Figure 3 - Relationship among the types

Figure 3 - Relationship among the types (continued)

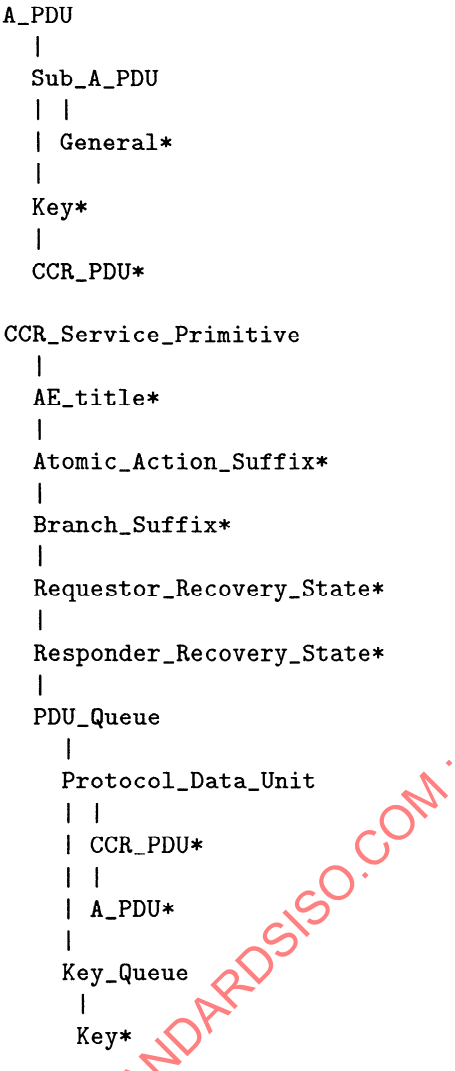

Figure 3 - Relationship among the types (continued)

Figure 3 - Relationship among the types (continued)

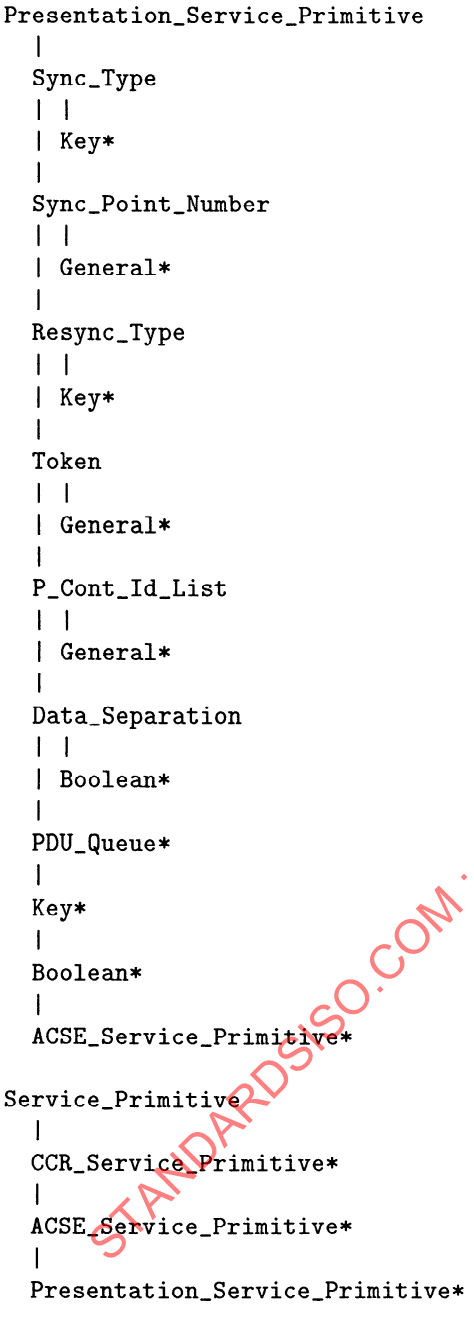

Figure 3 - Relationship among the types (continued)

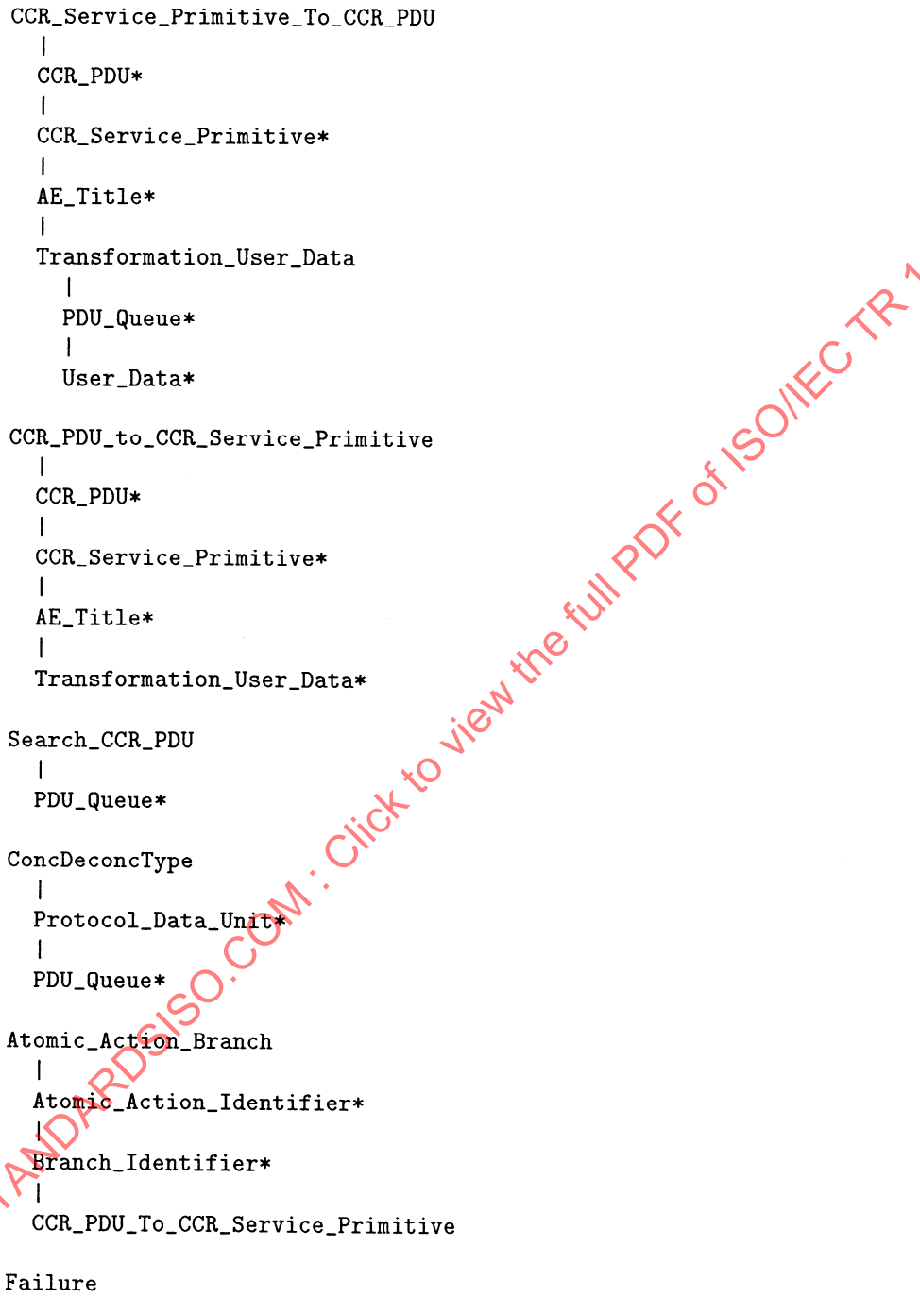

Figure 3 - Relationship among the types (concluded)

7 Interface of the description of CCR protocol

CCR protocol is represented in terms of type definitions and process definitions. The static attribute, i.e. the type of the service primitives and their parameters are described in type definitions. The behaviour, i.e. the protocol behaviours issuing/receiving procedures are described in process definitions.

Issuing and receiving of protocol data units are represented as the events at the gates.

The model employed in this Technical Report is shown in Figure 1.

Each gate is used as follows:

`u`	:	to communicate with the CCR user
`p`	:	to communicate with the presentation layer
`acse_u`	:	to communicate with ACSE (from ACSE to CCR)
`acse_l`	:	to communicate with ACSE (from CCR to ACSE)
`u_env`	:	to communicate with an environment of a CCR user, such information as major or minor synchronization points or tokens
`env`	:	to communicate with an ALS environment, especially for the failure information

The parameter ver is a CCR version number, which is a constant either `ver1`, `ver2`, `ver12`, or `rejected`.

Formal data types, `Boolean` and `Element`, are imported from the LOTOS standard library.

```
specification   CCR_Protocol_in_ALS[u,p,acse_u,acse_l,env,u_env]
                                                (ver:ccr_ver) : exit

library
        Boolean,Element
endlib
```

8 Type definitions

8.1 General Type

This type is a general type. There are some types which have a structure, but whose structure is not important in this LOTOS description. In such a case, this type is used as a place holder.

```
type    General is Boolean
sorts   general
opns
        null : -> general
        succ : general -> general
        _eq_,_ne_ : general,general -> Bool
eqns
forall  x,y:general
ofsort  Bool
        null eq null = true;
        null eq succ(x) = false;
        succ(x) eq null = false;
        succ(x) eq succ(y) = x eq y;
```

```
        x ne y = not(x eq y);
endtype (* General *)
```

8.1.1 Key Type

This type is used for identification which is necessary in other types.

```
type    Key is Boolean
sorts   key
opns
        0 : -> key
        succ : key -> key
        _eq_,_ne_,_le_ : key,key -> Bool
eqns
forall  x,x1,y,y1:key
ofsort  Bool
        0 eq 0 = true;
        0 eq succ(x) = false;
        succ(x) eq 0 = false;
        succ(x) eq succ(y) = x eq y;
        x ne y = not(x eq y);
        0 le 0 = true;
        0 le succ(x) = true;
        succ(x) le succ(y) = x le y;
        succ(x) le 0 = false;
endtype (* Key *)
```

8.1.2 Formal Identifier Type

This type is a formal type for the definition of "Atomic Action Identifier" and "Branch Identifier". These two types have the same structure. Therefore, this formal type is useful for the definitions as a template.

```
type    F_Name is Element renamedby
sortnames
        f_name for Element
endtype (* F_Name *)

type    F_Suffix is Element renamedby
sortnames
        f_suffix for Element
endtype (* F_Suffix *)

type    F_Identifier is F_Name,F_Suffix
sorts
        f_identifier
opns
        f_identifier : f_name,f_suffix -> f_identifier
        get_fn : f_identifier -> f_name
        get_fs : f_identifier -> f_suffix
        _eq_,_ne_ : f_identifier,f_identifier -> FBool
eqns
forall  fn,fn1:f_name,fs,fs1:f_suffix,f,f1:f_identifier
```

```
ofsort  f_name
        get_fn(f_identifier(fn,fs)) = fn;
ofsort  f_suffix
        get_fs(f_identifier(fn,fs)) = fs;
ofsort  FBool
        f eq f1 = (get_fn(f) eq get_fn(f1)) and (get_fs(f) eq get_fs(f1));
        f ne f1 = not(f eq f1);
endtype (* F_Identifier *)
```

8.1.3 Atomic Action Identifier

This type consists of the pair of "Masters Name" and "Atomic Action Suffix". Formal Identifier Type is used.

```
type    F_Atomic_Action_Identifier is F_Identifier renamedby
sortnames
        atomic_action_identifier for f_identifier
opnnames
        atomic_action_identifier for f_identifier
        get_mn for get_fn
        get_aas for get_fs
endtype (* F_Atomic_Action_Identifier *)

type    Atomic_Action_Identifier is F_Atomic_Action_Identifier actualizedby
                    Atomic_Action_Suffix,Masters_Name,Boolean using
sortnames
        masters_name for f_name
        atomic_action_suffix for f_suffix
        Bool for FBool
endtype (* Atomic_Action_Identifier *)
```

8.1.4 Branch Identifier

This type consists of the pair of "Superiors Name" and "Branch Suffix". Formal Identifier Type is used.

```
type    F_Branch_Identifier is F_Identifier renamedby
sortnames
        branch_identifier for f_identifier
opnnames
        branch_identifier for f_identifier
        get_sn for get_fn
        get_bs for get_fs
endtype (* F_Branch_Identifier *)

type    Branch_Identifier is F_Branch_Identifier actualizedby
                             Branch_Suffix,Superiors_Name,Boolean using
sortnames
        superiors_name for f_name
        branch_suffix for f_suffix
        Bool for FBool
endtype (* Branch_Identifier *)
```

8.1.5 Masters Name

This type is constructed by the renaming of Name Type. This type and Superiors Name Type have the same structure.

```
type    Masters_Name is Name renamedby
sortnames
        masters_name for name
opnnames
        masters_name for name
endtype (* Masters_Name *)
```

8.1.6 Superiors Name

This type is constructed from Name Type. This type and Masters Name Type have the same structures.

```
type    Superiors_Name is Name renamedby
sortnames
        superiors_name for name
opnnames
        superiors_name for name
endtype (* Superiors_Name *)
```

8.1.7 Name Type

This is a general Name Type, and is used for the definition of Masters Name and Superiors Name. This type is composed of AE Title and Side.

```
type    Name is AE_title,Side,Boolean,CCR_Version_Number
sorts   name
opns
        name : ae_title -> name
        name : side -> name
        get_aet : name -> ae_title
        get_s : name -> side
        is_aet : name -> Bool
        is_s : name -> Bool
        _eq_,_ne_: name,name -> Bool
        get_aet: name, ccr_ver, ae_title, ae_title -> ae_title
eqns
forall  aet,sender_aet,receiver_aet:    ae_title,
        s,s1:   side,
        n,n1:   name,
        ver:    ccr_ver
ofsort  Bool
        is_aet(name(aet)) = true;
        is_aet(name(s)) = false;
        is_s(name(aet)) = false;
        is_s(name(s)) = true;
ofsort  AE_title
        get_aet(name(aet)) = aet;
```

```
        get_aet(name(aet),ver,receiver_aet,sender_aet) = aet;
        get_aet(name(receiver),ver2,receiver_aet,sender_aet) = receiver_aet;
        get_aet(name(sender),ver2,receiver_aet,sender_aet) = sender_aet;
ofsort  side
        get_s(name(s)) =s;
ofsort  Bool
        is_aet(n) and is_aet(n1) =>
                n eq n1 = get_aet(n) eq get_aet(n1);
        is_s(n) and is_s(n1) =>
                n eq n1 = get_s(n) eq get_s(n1);
        is_aet(n) and is_s(n1) =>
                n eq n1 = false;
        is_s(n) and is_aet(n1) =>       n eq n1 = false;
        n ne n1 = not(n eq n1);
endtype (* Name *)
```

8.1.8 AE Title Type

AE Title is an important name in OSI Application Layer, which is composed of AP Title and AE Qualifier.

```
type    AE_title is AP_title,AE_qualifier,Boolean
sorts   ae_title
opns
        ae_null : -> ae_title
        aet : ap_title,ae_qualifier -> ae_title
        get_apt : ae_title -> ap_title
        get_aeq : ae_title -> ae_qualifier
        _eq_,_ne_ : ae_title,ae_title -> Bool
eqns
forall  apt,apt1:ap_title,aeq,aeq1:ae_qualifier,aet,aet1:ae_title
ofsort  ap_title
        get_apt(aet(apt,aeq)) = apt;
ofsort  ae_qualifier
        get_aeq(aet(apt,aeq)) = aeq;
ofsort  ae_title
        ae_null = aet(ap_null,aq_null);
ofsort  Bool
        aet(apt,aeq) eq aet(apt1,aeq1) = (apt eq apt1) and (aeq eq aeq1);
        aet ne aet1 = not(aet eq aet1);
endtype (* AE_title *)
```

8.1.9 AP Title Type

AP Title is defined in the Technical Corrigendum of OSI ACSE , which is Name defined by Directory Service or ASN1 OCTET IDENTIFIER. But, in this description, its structure is not important. AP Title is only used as a place holder.

```
type    AP_title is General renamedby
sortnames
        ap_title for general
opnnames
        ap_null for null
```

```
endtype (* AP_title *)
```

8.1.10 AE Qualifier Type

AP Title is defined in the Technical Corrigendum of OSI ACSE , which is RelativeDistingushedName defined by Directory Service or ASN1 INTEGER. But, in this description, its structure is not important. AP Title is only used as a place holder.

```
type    AE_qualifier is General renamedby
sortnames
        ae_qualifier for general
opnnames
        aq_null for null
endtype (* AE_qualifier *)
```

8.2 Side Type

Side is used in CCR version 2. It includes "sender" or "receiver". The notation of side depends on AE Titles which are transferred in the establishment of an association. It is used in CCR PDU.

```
type    Side is Boolean
sorts   side
opns
        sender,receiver : -> side
        _eq_,_ne_ : side,side -> Bool
eqns
        forall s1, s2 : side
ofsort  Bool
        sender eq sender = true;
        receiver eq sender = false;
        sender eq receiver = false;
        receiver eq receiver = true;
        s1 ne s2 = not(s1 eq s2);
endtype (* Side *)
```

8.2.1 Atomic Action Suffix Type

This type is a kind of identifier of Atomic Action, which is more local than Atomic Action Identifier. In this description, it is not important and is used as a place holder. It is used in CCR Service Primitive and CCR PDU.

```
type    Atomic_Action_Suffix is General renamedby
sortnames
        atomic_action_suffix for general
endtype (* Atomic_Action_Suffix *)
```

8.2.2 Branch Suffix Type

This type is a kind of identifier of Branch, which is more local than Branch Identifier. In this description, it is not important and is used as a place holder. It is used in CCR Service Primitive and CCR PDU.

```
type    Branch_Suffix is General renamedby
sortnames
        branch_suffix for general
endtype (* Branch_Suffix *)
```

8.2.3 Recovery State Type

This represents recovery state in CCR. In ASN.1, this type is represented by two ENUMERATE types. Therefore, in this description, two types, i.e., Requestor_Recovery_State and Responder_Recovery_State, are also defined. It is used in CCR Service Primitive and CCR PDU.

```
type    Requestor_Recovery_State is Key
sorts   requestor_recovery_state
opns
        commit,ready : -> requestor_recovery_state
        h : requestor_recovery_state -> key
        _eq_,_ne_ : requestor_recovery_state,requestor_recovery_state -> Bool
eqns
forall  r,r1:requestor_recovery_state
ofsort  key
        h(commit) = 0;
        h(ready) = succ(h(commit));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Requestor_Recovery_State *)

type    Responder_Recovery_State is Key
sorts   responder_recovery_state
opns
        done,unknown,retry : -> responder_recovery_state
        h : responder_recovery_state -> key
        _eq_,_ne_ : responder_recovery_state,responder_recovery_state -> Bool
eqns
forall  r,r1:responder_recovery_state
ofsort  key
        h(done) = 0;
        h(unknown) = succ(h(done));
        h(retry) = succ(h(unknown));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Responder_Recovery_State *)
```

8.2.4 Result

This is the result of an association establishment. It is used in ACSE Service Primitive.

```
type    Result is Key
sorts   result
opns
        accepted,rejected_permanent,rejected_transient : -> result
```

```
        _eq_,_ne_ : result,result -> Bool
        h : result -> key
eqns
forall  r,r1:result
ofsort  key
        h(accepted) = 0;
        h(rejected_permanent) = succ(h(accepted));
        h(rejected_transient) = succ(h(rejected_permanent));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Result *)
```

8.2.5 Result Source

This represents an entity (or a layer) which reports the result of an association establishment. It is used in ACSE Service Primitive.

```
type    Result_Source is Key
sorts   result_source
opns
        ACSE_service_user,ACSE_service_provider,presentation_service_provider
                : -> result_source
        _eq_,_ne_ : result_source,result_source -> Bool
        h : result_source -> key
eqns
forall  r,r1:result_source
ofsort  key
        h(ACSE_service_user) = 0;
        h(ACSE_service_provider) = succ(h(ACSE_service_user));
        h(presentation_service_provider) = succ(h(ACSE_service_provider));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* Result_Source *)
```

8.2.6 Diagnostic

This represents a diagnostic about the result of an association establishment. It is used in ACSE Service Primitive.

```
type    Diagnostic is
sorts   diagnostic
opns
        no_reason_given : -> diagnostic
endtype (* Diagnostic *)
```

8.2.7 Other Parameters

This is just a place holder, in which other parameters except for the above in ACSE Service Primitive are set.

```
type    Other_Parameters is General renamedby
sortnames
        other_parameters for general
endtype (* Other_Parameters *)
```

8.2.8 Synchronization Type

Synchronization Type is a type of minor synchronization in Session Layer. "Explicit" and "Optional" are included.

```
type    Sync_Type is Key
sorts   sync_type
opns
        exp,opt : -> sync_type
        h : sync_type -> key
        _eq_,_ne_ : sync_type,sync_type -> Bool
eqns
forall  s,s1:sync_type
ofsort  key
        h(exp) = 0;
        h(opt) = succ(h(exp));
ofsort  Bool
        s eq s1 = h(s) eq h(s1);
        s ne s1 = not(s eq s1);
endtype (* Sync_Type *)
```

8.2.9 Synchronization Point Serial Number

This type may be an integer actually, but it is not important in this specification. Therefore, it is represented by General Type.

```
type    Sync_Point_Number is General renamedby
sortnames
        sync_point for general
opnnames
        s_null for null
endtype (* Sync_Point_Number *)
```

8.2.10 Re-Synchronization Type

This type is the resynchronization type in Session Layer. Three terms, abandon, restart and set, are defined.

```
type    ReSync_Type is Key
sorts   resync_type
opns
        abandon,restart,set : -> resync_type
        h : resync_type -> key
        _eq_,_ne_ : resync_type,resync_type -> Bool
eqns
```

```
forall  r,r1:resync_type
ofsort  key
        h(abandon) = 0;
        h(restart) = succ(h(abandon));
        h(set) = succ(h(restart));
ofsort  Bool
        r eq r1 = h(r) eq h(r1);
        r ne r1 = not(r eq r1);
endtype (* ReSync_Type *)
```

8.2.11 Token Type

Token is used in Session Layers. Token is an important for CCR user, but unless the description of CCR user is detailed in a certain sense, it is difficult to describe that CCR provider checks whether the service requestor has the valid token or not. This description specifies the concerns about CCR providers. Therfore, this type is simply defined. When in this descrption token is necessary (e.g. mapping CCR PDUs to presentation services), the environment offers the specific values.

```
type    Token is General renamedby
sortnames
        token for general
opnnames
        t_null for null
endtype (* Token *)
```

8.2.12 Presentation Context Identifier List

This type is not important in this specification.

```
type    P_Cont_Id_List is General renamedby
sortnames
        p_cont_id_list for general
opnnames
        p_null for null
endtype (* P_Cont_Id_List *)
```

8.2.13 Data Separation Type

Data Separation is one of Session Functional Units. It is only used in CCR version 2.

```
type    Data_Separation is Boolean renamedby
sortnames
        data_sep for Bool
opnnames
        on for true
        off for false
endtype (* Data_Separation *)
```

8.2.14 CCR Protocol Data Unit User Data

This type is not important in this specification, and a just place holder.

```
type    User_Data is General renamedby
sortnames
        user_data for general
endtype (* User_Data *)
```

8.2.15 Transformation Between User_Data and CCR_User_Data

In this description, the user data of each service Primitive is represented by PDU_Queue. It is necessary for the concatenation/deconcatenation of PDUs for Application Layers in Presentation Layer Service Primitives. The type of user data of service primitives are different from that of CCR PDUs. Therefore, the transformation functions between two different types are necessary.

```
type    Transformation_User_Data is PDU_Queue,User_Data
opns
        to_pdu: PDUqueue -> user_data
        to_sp : user_data -> PDUqueue
eqns
forall  q:PDUqueue,ud:user_data
ofsort  PDUqueue
        to_sp(to_pdu(q)) = q;
ofsort  user_data
        to_pdu(to_sp(ud)) = ud;
endtype (* Transformation_User_Data *)
```

9 Service Primitive Definitions

Three types of Service Primitives are necessary. CCR Service Primitives, ACSE Service Primitives, and Presentation Layer Service Primitives.

9.1 CCR Service Primitives Type

This type corresponds the service primitives of CCR.

```
type    CCR_Service_Primitive is AE_title,Atomic_Action_Suffix,
                                 Branch_Suffix,
                                 Requestor_Recovery_State,
                                 Responder_Recovery_State,
                                 PDU_Queue
sorts   CSP
opns
        CBeginReq : ae_title (* Atomic Action Identifier Masters Name *),
                    atomic_action_suffix (* Atomic Action Identifier Suffix *),
                    ae_title (* Branch Identifier Superiors Name *),
                    branch_suffix (* Branch Identifier Suffix *),
                    PDUqueue (* User Data *) -> CSP
```

```
CBeginInd : ae_title (* Atomic Action Identifier Masters Name *),
            atomic_action_suffix (* Atomic Action Identifier Suffix *),
            ae_title (* Branch Identifier Superiors Name *),
            branch_suffix (* Branch Identifier Suffix *),
            PDUqueue (* User Data *) -> CSP
CBeginRsp : PDUqueue (* User Data *) -> CSP
CBeginCnf : PDUqueue (* User Data *) -> CSP
CPrepareReq : PDUqueue (* User Data *) -> CSP
CPrepareInd : PDUqueue (* User Data *) -> CSP
CReadyReq : PDUqueue (* User Data *) -> CSP
CReadyInd : PDUqueue (* User Data *) -> CSP
CCommitReq : PDUqueue (* User Data *) -> CSP
CCommitInd : PDUqueue (* User Data *) -> CSP
CCommitRsp : PDUqueue (* User Data *) -> CSP
CCommitCnf : PDUqueue (* User Data *) -> CSP
CRollbackReq : PDUqueue (* User Data *) -> CSP
CRollbackInd : PDUqueue (* User Data *) -> CSP
CRollbackRsp : PDUqueue (* User Data *) -> CSP
CRollbackCnf : PDUqueue (* User Data *) -> CSP
CRecoverReq : requestor_recovery_state,
              ae_title (* Atomic Action Identifier Masters Name *),
              atomic_action_suffix (* Atomic Action Identifier
                                      Suffix *),
              ae_title (* Branch Identifier Superiors Name *),
              branch_suffix (* Branch Identifier Suffix *),
              PDUqueue (* User Data *) -> CSP
CRecoverInd : requestor_recovery_state,
              ae_title (* Atomic Action Identifier Masters Name *),
              atomic_action_suffix (* Atomic Action Identifier
                                      Suffix *),
              ae_title (* Branch Identifier Superiors Name *),
              branch_suffix (* Branch Identifier Suffix *),
              PDUqueue (* User Data *) -> CSP
CRecoverRsp : responder_recovery_state,
              ae_title (* Atomic Action Identifier Masters Name *),
              atomic_action_suffix (* Atomic Action Identifier
                                      Suffix *),
              ae_title (* Branch Identifier Superiors Name *),
              branch_suffix (* Branch Identifier Suffix *),
              PDUqueue (* User Data *) -> CSP
CRecoverCnf : responder_recovery_state,
              ae_title (* Atomic Action Identifier Masters Name *),
              atomic_action_suffix (* Atomic Action Identifier
                                      Suffix *),
              ae_title (* Branch Identifier Superiors Name *),
              branch_suffix (* Branch Identifier Suffix *),
              PDUqueue (* User Data *) -> CSP
CCommitReq_CBeginReq : PDUqueue (* User Data *),
                       ae_title (* Atomic Action Identifier Masters
                                   Name *),
                       atomic_action_suffix (* Atomic Action Identifier
                                               Suffix *),
                       ae_title (* Branch Identifier Superiors Name *),
                       branch_suffix (* Branch Identifier Suffix *),
```

```
                              PDUqueue (* User Data *) -> CSP
        CRollbackReq_CBeginReq : PDUqueue (* User Data *),
                                 ae_title (* Atomic Action Identifier Masters
                                             Name *),
                                 atomic_action_suffix (* Atomic Action Identifier
                                                         Suffix *),
                                 ae_title (* Branch Identifier Superiors Name *),
                                 branch_suffix (* Branch Identifier Suffix *),
                                 PDUqueue (* User Data *) -> CSP

        CBegin_req,CBegin_ind,CBegin_rsp,CBegin_cnf,CPrepare_req,CPrepare_ind,
        CReady_req,CReady_ind,CCommit_req,CCommit_ind,CCommit_rsp,CCommit_cnf,
        CRollback_req,CRollback_ind,CRollback_rsp,CRollback_cnf,
        CRecover_req,CRecover_ind,CRecover_rsp,CRecover_cnf,
        CCommit_req_CBegin_req,CRollback_req_CBegin_req : -> key

        is_CBeginReq,is_CBeginInd,is_CBeginRsp,is_CBeginCnf,
        is_CPrepareReq,is_CPrepareInd,is_CReadyReq,is_CReadyind,
        is_CCommitReq,is_CCommitInd,is_CCommitRsp,is_CCommitCnf,
        is_CRollbackReq,is_CRollbackInd,is_CRollbackRsp,is_CRollbackCnf,
        is_CRecoverReq,is_CRecoverInd,is_CRecoverRsp,is_CRecoverCnf,
        is_CCommitReq_CBeginReq,is_CRollbackReq_CBeginReq : CSP -> bool

        get_aaimn : CSP -> ae_title (* Atomic Action Identifier Masters Name *)
        get_aais : CSP -> atomic_action_suffix (* Atomic Action Identifier
                                                  Suffix *)
        get_bisn : CSP -> ae_title (* Branch Identifier Superiors Name *)
        get_bis : CSP -> branch_suffix (* Branch Identifier Suffix *)
        get_ud : CSP -> PDUqueue (* User Data *)
        get_udb : CSP -> PDUqueue (* User Data *)
        get_rs : CSP -> requestor_recovery_state
        get_rr : CSP -> responder_recovery_state

        h : CSP -> key
eqns
forall  aaimn:ae_title (* Atomic Action Identifier Masters Name *),
        aais:atomic_action_suffix (* Atomic Actiona Identifier Suffix *),
        bisn:ae_title (* Branch Identifier Superiors Name *),
        bis:branch_suffix (* Branch Identifier Suffix *),
        ud,udb:PDUqueue (* User Data *),
        rs:requestor_recovery_state,rr:responder_recovery_state,csp:CSP
ofsort  key
        CBegin_req = 0;
        CBegin_ind = succ(CBegin_req);
        CBegin_rsp = succ(CBegin_ind);
        CBegin_cnf = succ(CBegin_rsp);
        CPrepare_req = succ(CBegin_cnf);
        CPrepare_ind = succ(CPrepare_req);
        CReady_req = succ(CPrepare_ind);
        CReady_ind = succ(CReady_req);
        CCommit_req = succ(CReady_ind);
        CCommit_ind = succ(CCommit_req);
        CCommit_rsp = succ(CCommit_ind);
        CCommit_cnf = succ(CCommit_rsp);
```

```
        CRollback_req = succ(CCommit_cnf);
        CRollback_ind = succ(CRollback_req);
        CRollback_rsp = succ(CRollback_ind);
        CRollback_cnf = succ(CRollback_rsp);
        CRecover_req = succ(CRollback_cnf);
        CRecover_ind = succ(CRecover_req);
        CRecover_rsp = succ(CRecover_ind);
        CRecover_cnf = succ(CRecover_rsp);
        CCommit_req_CBegin_req = succ(CRecover_cnf);
        CRollback_req_CBegin_req = succ(CCommit_req_CBegin_req);

        h(CBeginReq(aaimn,aais,bisn,bis,ud)) = CBegin_req;
        h(CBeginInd(aaimn,aais,bisn,bis,ud)) = CBegin_ind;
        h(CBeginRsp(ud)) = CBegin_rsp;
        h(CBeginCnf(ud)) = CBegin_cnf;
        h(CPrepareReq(ud)) = CPrepare_req;
        h(CPrepareInd(ud)) = CPrepare_ind;
        h(CReadyReq(ud)) = CReady_req;
        h(CReadyInd(ud)) = CReady_ind;
        h(CCommitReq(ud)) = CCommit_req;
        h(CCommitInd(ud)) = CCommit_ind;
        h(CCommitRsp(ud)) = CCommit_rsp;
        h(CCommitCnf(ud)) = CCommit_cnf;
        h(CRollbackReq(ud)) = CRollback_req;
        h(CRollbackInd(ud)) = CRollback_ind;
        h(CRollbackRsp(ud)) = CRollback_rsp;
        h(CRollbackCnf(ud)) = CRollback_cnf;
        h(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = CRecover_req;
        h(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = CRecover_ind;
        h(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = CRecover_rsp;
        h(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = CRecover_cnf;
        h(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb))
                = CCommit_req_CBegin_req;
        h(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb))
                = CRollback_req_CBegin_req ;
ofsort bool
        is_CBeginReq(csp) = h(csp) eq CBegin_req;
        is_CBeginInd(csp) = h(csp) eq CBegin_ind;
        is_CBeginRsp(csp) = h(csp) eq CBegin_rsp;
        is_CBeginCnf(csp) = h(csp) eq CBegin_cnf;
        is_CPrepareReq(csp) = h(csp) eq CPrepare_req;
        is_CPrepareInd(csp) = h(csp) eq CPrepare_ind;
        is_CReadyReq(csp) = h(csp) eq CReady_req;
        is_CReadyInd(csp) = h(csp) eq CReady_ind;
        is_CCommitReq(csp) = h(csp) eq CCommit_req;
        is_CCommitInd(csp) = h(csp) eq CCommit_ind;
        is_CCommitRsp(csp) = h(csp) eq CCommit_rsp;
        is_CCommitCnf(csp) = h(csp) eq CCommit_cnf;
        is_CRollbackReq(csp) = h(csp) eq CRollback_req;
        is_CRollbackInd(csp) = h(csp) eq CRollback_ind;
        is_CRollbackRsp(csp) = h(csp) eq CRollback_rsp;
        is_CRollbackCnf(csp) = h(csp) eq CRollback_cnf;
        is_CRecoverReq(csp) = h(csp) eq CRecover_req;
        is_CRecoverInd(csp) = h(csp) eq CRecover_ind;
```

```
        is_CRecoverRsp(csp) = h(csp) eq CRecover_rsp;
        is_CRecoverCnf(csp) = h(csp) eq CRecover_cnf;
        is_CCommitReq_CBeginReq(csp) = h(csp) eq CCommit_req_CBegin_req;
        is_CRollbackReq_CBeginReq(csp) = h(csp) eq CRollback_req_CBegin_req;
ofsort  ae_title (* Atomic Action Identifier Masters Name *)
        get_aaimn(CBeginReq(aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CBeginInd(aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = aaimn;
        get_aaimn(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aaimn;
        get_aaimn(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aaimn;
ofsort  atomic_action_suffix (* Atomic Action Identifier Suffix *)
        get_aais(CBeginReq(aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CBeginInd(aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = aais;
        get_aais(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aais;
        get_aais(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = aais;
ofsort  ae_title (* Branch Identifier Superiors Name *)
        get_bisn(CBeginReq(aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CBeginInd(aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = bisn;
        get_bisn(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bisn;
        get_bisn(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bisn;
ofsort  branch_suffix (* Branch Identifier Suffix *)
        get_bis(CBeginReq(aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CBeginInd(aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = bis;
        get_bis(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bis;
        get_bis(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = bis;
ofsort  PDUqueue (* User Data *)
        get_ud(CBeginReq(aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CBeginInd(aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CBeginRsp(ud)) = ud;
        get_ud(CBeginCnf(ud)) = ud;
        get_ud(CPrepareReq(ud)) = ud;
        get_ud(CPrepareInd(ud)) = ud;
        get_ud(CReadyReq(ud)) = ud;
        get_ud(CReadyInd(ud)) = ud;
        get_ud(CCommitReq(ud)) = ud;
        get_ud(CCommitInd(ud)) = ud;
        get_ud(CCommitRsp(ud)) = ud;
        get_ud(CCommitCnf(ud)) = ud;
        get_ud(CRollbackReq(ud)) = ud;
```

```
        get_ud(CRollbackInd(ud)) = ud;
        get_ud(CRollbackRsp(ud)) = ud;
        get_ud(CRollbackCnf(ud)) = ud;
        get_ud(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = ud;
        get_ud(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
        get_ud(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = ud;
        get_udb(CCommitReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = udb;
        get_udb(CRollbackReq_CBeginReq(ud,aaimn,aais,bisn,bis,udb)) = udb;
ofsort  requestor_recovery_state
        get_rs(CRecoverReq(rs,aaimn,aais,bisn,bis,ud)) = rs;
        get_rs(CRecoverInd(rs,aaimn,aais,bisn,bis,ud)) = rs;
ofsort  responder_recovery_state
        get_rr(CRecoverRsp(rr,aaimn,aais,bisn,bis,ud)) = rr;
        get_rr(CRecoverCnf(rr,aaimn,aais,bisn,bis,ud)) = rr;
endtype (* CCR_Service_Primitive *)
```

9.2 ACSE Service Primitives Type

This Definition does not contain all ACSE Services, but ones which are necessary for the description of CCR Protocol Machine are only defined.

```
type    ACSE_Service_Primitive is AP_title,AE_Qualifier,
                                  Result,Result_Source,Diagnostic,
                                  PDU_Queue,Other_Parameters,Boolean,
                                  CCR_Service_Primitive
sorts   ASP
opns
        AAssocReq : ap_title (* Calling AP Title *),
                    ae_qualifier (* Calling AE Qualifier *),
                    ap_title (* Called AP Title *),
                    ae_qualifier (* Called AE Qualifier *),
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocInd : ap_title (* Calling AP Title *),
                    ae_qualifier (* Calling AE Qualifier *),
                    ap_title (* Called AP Title *),
                    ae_qualifier (* Called AE Qualifier *),
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocRsp : ap_title (* Responding AP Title *),
                    ae_qualifier (* Responding AE Qualifier *),
                    result,result_source,diagnostic,
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
        AAssocCnf : ap_title (* Responding AP Title *),
                    ae_qualifier (* Responding AE Qualifier *),
                    result,result_source,diagnostic,
                    PDUqueue (* User Data *),
                    other_parameters -> ASP
```

```
        AAssoc_req,AAssoc_ind,AAssoc_rsp,AAssoc_cnf : -> key

        is_AAssocReq,is_AAssocInd,is_AAssocRsp,is_AAssocCnf : ASP -> bool

        get_cgat : ASP -> ap_title (* Calling AP Title *)
        get_cgaq : ASP -> ae_qualifier (* Calling AE Qualifier *)
        get_cdat : ASP -> ap_title (* Called AP Title *)
        get_cdaq : ASP -> ae_qualifier (* Called AE Qualifier *)
        get_rat : ASP -> ap_title (* Responding AP Title *)
        get_raq : ASP -> ae_qualifier (* Responding AE Qualifier *)
        get_r : ASP -> result
        get_rs : ASP -> result_source
        get_d : ASP -> diagnostic
        get_ud : ASP -> PDUqueue (* User Data *)
        get_op : ASP -> other_parameters

        h : ASP -> key
eqns
forall  cgat:ap_title (* Calling AP Title *),
        cgaq:ae_qualifier (* Calling AE Qualifier *),
        cdat:ap_title (* Called AP Title *),
        cdaq:ae_qualifier (* Called AE Qualifier *),
        rat:ap_title (* Responding AP Title *),
        raq:ae_qualifier (* Responding AE Qualifier *),
        r:result,rs:result_source,d:diagnostic,
        ud:PDUqueue (* User Data *),
        op:other_parameters,asp:ASP
ofsort  key
        AAssoc_req = succ(CRollback_req_CBegin_req);
        AAssoc_ind = succ(AAssoc_req);
        AAssoc_rsp = succ(AAssoc_ind);
        AAssoc_cnf = succ(AAssoc_rsp);

        h(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = AAssoc_req;
        h(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = AAssoc_ind;
        h(AAssocRsp(rat,raq,r,rs,d,ud,op)) = AAssoc_rsp;
        h(AAssocCnf(rat,raq,r,rs,d,ud,op)) = AAssoc_cnf;
ofsort  bool
        is_AAssocReq(asp) = h(asp) eq AAssoc_req;
        is_AAssocInd(asp) = h(asp) eq AAssoc_ind;
        is_AAssocRsp(asp) = h(asp) eq AAssoc_rsp;
        is_AAssocCnf(asp) = h(asp) eq AAssoc_cnf;
ofsort  ap_title (* Calling AP Title *)
        get_cgat(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cgat;
        get_cgat(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cgat;
ofsort  ae_qualifier (* Calling AE Qualifier *)
        get_cgaq(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cgaq;
        get_cgaq(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cgaq;
ofsort  ap_title (* Called AP Title *)
        get_cdat(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cdat;
        get_cdat(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cdat;
ofsort  ae_qualifier (* Called AE Qualifier *)
        get_cdaq(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = cdaq;
        get_cdaq(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = cdaq;
```

```
ofsort  ap_title (* Responding AP Title *)
        get_rat(AAssocRsp(rat,raq,r,rs,d,ud,op)) = rat;
        get_rat(AAssocCnf(rat,raq,r,rs,d,ud,op)) = rat;
ofsort  ae_qualifier (* Responding AE Qualifier *)
        get_raq(AAssocRsp(rat,raq,r,rs,d,ud,op)) = raq;
        get_raq(AAssocCnf(rat,raq,r,rs,d,ud,op)) = raq;
ofsort  result
        get_r(AAssocRsp(rat,raq,r,rs,d,ud,op)) = r;
        get_r(AAssocCnf(rat,raq,r,rs,d,ud,op)) = r;
ofsort  result_source
        get_rs(AAssocRsp(rat,raq,r,rs,d,ud,op)) = rs;
        get_rs(AAssocCnf(rat,raq,r,rs,d,ud,op)) = rs;
ofsort  diagnostic
        get_d(AAssocRsp(rat,raq,r,rs,d,ud,op)) = d;
        get_d(AAssocCnf(rat,raq,r,rs,d,ud,op)) = d;
ofsort  PDUqueue (* User Data *)
        get_ud(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = ud;
        get_ud(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = ud;
        get_ud(AAssocRsp(rat,raq,r,rs,d,ud,op)) = ud;
        get_ud(AAssocCnf(rat,raq,r,rs,d,ud,op)) = ud;
ofsort  other_parameters
        get_op(AAssocReq(cgat,cgaq,cdat,cdaq,ud,op)) = op;
        get_op(AAssocInd(cgat,cgaq,cdat,cdaq,ud,op)) = op;
        get_op(AAssocRsp(rat,raq,r,rs,d,ud,op)) = op;
        get_op(AAssocCnf(rat,raq,r,rs,d,ud,op)) = op;
endtype (* ACSE_Service_Primitive *)
```

9.3 Presentation Service Primitive Definition

This Definition does not contain all Presentation Services, but only the ones which are necessary for the description of CCR Protocol Machine are defined.

```
type    Presentation_Service_Primitive is Sync_Type,Sync_Point_Number,
                                          Resync_Type,Token,
                                          P_Cont_Id_List,Data_Separation,
                                          PDU_Queue,Key,Boolean,
                                          ACSE_Service_Primitive
sorts   PSP
opns
        PConnectInd : -> PSP
        PConnectCnf : -> PSP

        PSyncMinReq : sync_type,sync_point,PDUqueue (* User Data *),
                      data_sep -> PSP
        PSyncMinInd : sync_type,sync_point,PDUqueue (* User Data *),
                      data_sep -> PSP
        PSyncMinRsp : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMinCnf : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajReq : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajInd : sync_point,PDUqueue (* User Data *) -> PSP
        PSyncMajRsp : PDUqueue (* User Data *) -> PSP
        PSyncMajCnf : PDUqueue (* User Data *) -> PSP
        PReSyncReq : resync_type,sync_point,token,
```

```
                PDUqueue (* User Data *) -> PSP
        PReSyncInd : resync_type,sync_point,token,p_cont_id_list,
                PDUqueue (* User Data *) -> PSP
        PReSyncRsp : sync_point,token,p_cont_id_list,
                PDUqueue (* User Data *) -> PSP
        PReSyncCnf : sync_point,token,PDUqueue (* User Data *) -> PSP
        PTypedDataReq : PDUqueue (* User Data *) -> PSP
        PTypedDataInd : PDUqueue (* User Data *) -> PSP

        PConnect_ind,PConnect_cnf,
        PSyncMin_req,PSyncMin_ind,PSyncMin_rsp,PSyncMin_cnf,
        PSyncMaj_req,PSyncMaj_ind,PSyncMaj_rsp,PSyncMaj_cnf,
        PReSync_req,PReSync_ind,PReSync_rsp,PReSync_cnf,
        PTypedData_req,PTypedData_ind : -> key

        is_PConnectInd,is_PConnectCnf,
        is_PSyncMinReq,is_PSyncMinInd,is_PSyncMinRsp,is_PSyncMinCnf,
        is_PSyncMajReq,is_PSyncMajInd,is_PSyncMajRsp,is_PSyncMajCnf,
        is_PReSyncReq,is_PReSyncInd,is_PReSyncRsp,is_PReSyncCnf,
        is_PTypedDataReq,is_PTypedDataInd,is_PSP_for_ACSE : PSP -> Bool

        get_st : PSP -> sync_type
        get_sp : PSP -> sync_point
        get_rt : PSP -> resync_type
        get_t : PSP -> token
        get_pcil : PSP -> p_cont_id_list
        get_ud : PSP -> PDUqueue (* User Data *)

        h : PSP -> key
eqns
forall  st:sync_type,sp:sync_point,rt:resync_type,t:token,pcil:p_cont_id_list,
        ud:PDUqueue (* User Data *),ds:data_sep,psp:PSP
ofsort  key
        PConnect_ind = succ(AAssoc_cnf);
        PConnect_cnf = succ(PConnect_ind);
        PSyncMin_req = succ(PConnect_cnf);
        PSyncMin_ind = succ(PSyncMin_req);
        PSyncMin_rsp = succ(PSyncMin_ind);
        PSyncMin_cnf = succ(PSyncMin_rsp);
        PSyncMaj_req = succ(PSyncMin_cnf);
        PSyncMaj_ind = succ(PSyncMaj_req);
        PSyncMaj_rsp = succ(PSyncMaj_ind);
        PSyncMaj_cnf = succ(PSyncMaj_rsp);
        PReSync_req = succ(PSyncMaj_cnf);
        PReSync_ind = succ(PReSync_req);
        PReSync_rsp = succ(PReSync_ind);
        PReSync_cnf = succ(PReSync_rsp);
        PTypedData_req = succ(PReSync_cnf);
        PTypedData_ind = succ(PTypedData_req);

        h(PConnectInd) = PConnect_ind;
        h(PConnectCnf) = PConnect_cnf;
        h(PSyncMinReq(st,sp,ud,ds)) = PSyncMin_req;
        h(PSyncMinInd(st,sp,ud,ds)) = PSyncMin_ind;
```

```
        h(PSyncMinRsp(sp,ud)) = PSyncMin_rsp;
        h(PSyncMinCnf(sp,ud)) = PSyncMin_cnf;
        h(PSyncMajReq(sp,ud)) = PSyncMaj_req;
        h(PSyncMajInd(sp,ud)) = PSyncMaj_ind;
        h(PSyncMajRsp(ud)) = PSyncMaj_rsp;
        h(PSyncMajCnf(ud)) = PSyncMaj_cnf;
        h(PReSyncReq(rt,sp,t,ud)) = PReSync_req;
        h(PReSyncInd(rt,sp,t,pcil,ud)) = PReSync_ind;
        h(PReSyncRsp(sp,t,pcil,ud)) = PReSync_rsp;
        h(PReSyncCnf(sp,t,ud)) = PReSync_cnf;
        h(PTypedDataReq(ud)) = PTypedData_req;
        h(PTypedDataInd(ud)) = PTypedData_ind;
ofsort  Bool
        is_PConnectInd(psp) = h(psp) eq PConnect_ind;
        is_PConnectCnf(psp) = h(psp) eq PConnect_cnf;
        is_PSyncMinReq(psp) = h(psp) eq PSyncMin_req;
        is_PSyncMinInd(psp) = h(psp) eq PSyncMin_ind;
        is_PSyncMinRsp(psp) = h(psp) eq PSyncMin_rsp;
        is_PSyncMinCnf(psp) = h(psp) eq PSyncMin_cnf;
        is_PSyncMajReq(psp) = h(psp) eq PSyncMaj_req;
        is_PSyncMajInd(psp) = h(psp) eq PSyncMaj_ind;
        is_PSyncMajRsp(psp) = h(psp) eq PSyncMaj_rsp;
        is_PSyncMajCnf(psp) = h(psp) eq PSyncMaj_cnf;
        is_PReSyncReq(psp) = h(psp) eq PReSync_req;
        is_PReSyncInd(psp) = h(psp) eq PReSync_ind;
        is_PReSyncRsp(psp) = h(psp) eq PReSync_rsp;
        is_PReSyncCnf(psp) = h(psp) eq PReSync_cnf;
        is_PTypedDataReq(psp) = h(psp) eq PTypedData_req;
        is_PTypedDataInd(psp) = h(psp) eq PTypedData_ind;
        is_PSP_for_ACSE(psp) = is_PConnectInd(psp) or is_PConnectCnf(psp);
ofsort  sync_type
        get_st(PSyncMinReq(st,sp,ud,ds)) = st;
        get_st(PSyncMinInd(st,sp,ud,ds)) = st;
ofsort  sync_point
        get_sp(PSyncMinReq(st,sp,ud,ds)) = sp;
        get_sp(PSyncMinInd(st,sp,ud,ds)) = sp;
        get_sp(PSyncMinRsp(sp,ud)) = sp;
        get_sp(PSyncMinCnf(sp,ud)) = sp;
        get_sp(PSyncMajReq(sp,ud)) = sp;
        get_sp(PSyncMajInd(sp,ud)) = sp;
        get_sp(PReSyncReq(rt,sp,t,ud)) = sp;
        get_sp(PReSyncInd(rt,sp,t,pcil,ud)) = sp;
        get_sp(PReSyncRsp(sp,t,pcil,ud)) = sp;
        get_sp(PReSyncCnf(sp,t,ud)) = sp;
ofsort  resync_type
        get_rt(PReSyncReq(rt,sp,t,ud)) = rt;
        get_rt(PReSyncInd(rt,sp,t,pcil,ud)) = rt;
ofsort  token
        get_t(PReSyncReq(rt,sp,t,ud)) = t;
        get_t(PReSyncInd(rt,sp,t,pcil,ud)) = t;
        get_t(PReSyncRsp(sp,t,pcil,ud)) = t;
        get_t(PReSyncCnf(sp,t,ud)) = t;
ofsort  p_cont_id_list
        get_pcil(PReSyncInd(rt,sp,t,pcil,ud)) = pcil;
```

```
        get_pcil(PReSyncRsp(sp,t,pcil,ud)) = pcil;
ofsort  PDUqueue (* User Data *)
        get_ud(PSyncMinReq(st,sp,ud,ds)) = ud;
        get_ud(PSyncMinInd(st,sp,ud,ds)) = ud;
        get_ud(PSyncMinRsp(sp,ud)) = ud;
        get_ud(PSyncMinCnf(sp,ud)) = ud;
        get_ud(PSyncMajReq(sp,ud)) = ud;
        get_ud(PSyncMajInd(sp,ud)) = ud;
        get_ud(PSyncMajRsp(ud)) = ud;
        get_ud(PSyncMajCnf(ud)) = ud;
        get_ud(PTypedDataReq(ud)) = ud;
        get_ud(PTypedDataInd(ud)) = ud;
        get_ud(PReSyncReq(rt,sp,t,ud)) = ud;
        get_ud(PReSyncInd(rt,sp,t,pcil,ud)) = ud;
        get_ud(PReSyncRsp(sp,t,pcil,ud)) = ud;
        get_ud(PReSyncCnf(sp,t,ud)) = ud;
endtype (* Presentation_Service_Primitive *)
```

9.4 General Service Primitive Type

This is a general service primitive type which represents Service Primitives in Application Layer and Presentation Layer. The type for general service primitivies is constructed from the Service Primitive types of ASEs which are CCR and ACSE and that of Presentation Layer.

```
type    Service_Primitive is CCR_Service_Primitive,
                             ACSE_Service_Primitive,
                             Presentation_Service_Primitive
sorts   SP
opns
        sp : CSP -> SP
        sp : ASP -> SP
        sp : PSP -> SP
        is_ccr_sp : SP -> Bool
        is_acse_sp : SP -> Bool
        is_pl_sp : SP -> Bool
        get_csp : SP -> CSP
        get_asp : SP -> ASP
        get_psp : SP -> PSP
eqns
forall  c:CSP,a:ASP,p:PSP
ofsort  CSP
        get_csp(sp(c)) = c;
ofsort  ASP
        get_asp(sp(a)) = a;
ofsort  PSP
        get_psp(sp(p)) = p;
ofsort  Bool
        is_ccr_sp(sp(c)) = true;
        is_ccr_sp(sp(a)) = false;
        is_ccr_sp(sp(p)) = false;

        is_acse_sp(sp(c)) = false;
        is_acse_sp(sp(a)) = true;
```

```
        is_acse_sp(sp(p)) = false;

        is_pl_sp(sp(c)) = false;
        is_pl_sp(sp(a)) = false;
        is_pl_sp(sp(p)) = true;
endtype (* Service_Primitive *)
```

10 PDU Types

In this description, two PDU types are defined. The first is CCR PDU and the second is other ASE PDU. The former is semantically important and the latter is a just place holder. But these PDUs are necessary in the discussion of concatenation/deconcatenation of PDUs.

10.1 CCR PDU Type

```
type    CCR_PDU is CCR_Version_Number,Atomic_Action_Identifier,
                   Branch_Identifier,Branch_Suffix,
                   Requestor_Recovery_State,Responder_Recovery_State,
                   User_Data,Key
sorts   CPDU
opns    c_initialize_ri : ccr_ver -> CPDU
        c_initialize_rc : ccr_ver -> CPDU
        c_begin_ri : atomic_action_identifier,
                     branch_suffix,
                     user_data -> CPDU
        c_begin_rc : user_data -> CPDU
        c_prepare_ri : user_data -> CPDU
        c_ready_ri : user_data -> CPDU
        c_commit_ri : user_data -> CPDU
        c_commit_rc : user_data -> CPDU
        c_rollback_ri : user_data -> CPDU
        c_rollback_rc : user_data -> CPDU
        c_recover_ri : atomic_action_identifier,
                       branch_identifier,
                       requestor_recovery_state,
                       user_data -> CPDU
        c_recover_rc : atomic_action_identifier,
                       branch_identifier,
                       responder_recovery_state,
                       user_data -> CPDU

        c_empty : -> CPDU

        c_initialize_ri,c_initialize_rc,c_begin_ri,c_begin_rc,
        c_prepare_ri,c_ready_ri,c_commit_ri,c_commit_rc,
        c_rollback_ri,c_rollback_rc,c_recover_ri,c_recover_rc : -> key

        is_c_initialize_ri,is_c_initialize_rc,
        is_c_begin_ri,is_c_begin_rc,is_c_prepare_ri,is_c_ready_ri,
        is_c_commit_ri,is_c_commit_rc,is_c_rollback_ri,is_c_rollback_rc,
        is_c_recover_ri,is_c_recover_rc : CPDU -> Bool
```

```
        get_v : CPDU -> ccr_ver
        get_aai : CPDU -> atomic_action_identifier
        get_bi : CPDU -> branch_identifier
        get_bs : CPDU -> branch_suffix
        get_ud : CPDU -> user_data
        get_rs : CPDU -> requestor_recovery_state
        get_rr : CPDU -> responder_recovery_state

        h : CPDU -> key
eqns
forall  v:ccr_ver,aai:atomic_action_identifier,bs:branch_suffix,
        bi:branch_identifier,ud,udb:user_data,rs:requestor_recovery_state,
        rr:responder_recovery_state,cpdu:CPDU
ofsort  key
        c_initialize_ri = 0;
        c_initialize_rc = succ(c_initialize_ri);
        c_begin_ri = succ(c_initialize_rc);
        c_begin_rc = succ(c_begin_ri);
        c_prepare_ri = succ(c_begin_rc);
        c_ready_ri = succ(c_prepare_ri);
        c_commit_ri = succ(c_ready_ri);
        c_commit_rc = succ(c_commit_ri);
        c_rollback_ri = succ(c_commit_rc);
        c_rollback_rc = succ(c_rollback_ri);
        c_recover_ri = succ(c_rollback_rc);
        c_recover_rc = succ(c_recover_ri);

        h(c_initialize_ri(v)) = c_initialize_ri;
        h(c_initialize_rc(v)) = c_initialize_rc;
        h(c_begin_ri(aai,bs,ud)) = c_begin_ri;
        h(c_begin_rc(ud)) = c_begin_rc;
        h(c_prepare_ri(ud)) = c_prepare_ri;
        h(c_ready_ri(ud)) = c_ready_ri;
        h(c_commit_ri(ud)) = c_commit_ri;
        h(c_commit_rc(ud)) = c_commit_rc;
        h(c_rollback_ri(ud)) = c_rollback_ri;
        h(c_rollback_rc(ud)) = c_rollback_rc;
        h(c_recover_ri(aai,bi,rs,ud)) = c_recover_ri;
        h(c_recover_rc(aai,bi,rr,ud)) = c_recover_rc;
ofsort  bool
        is_c_initialize_ri(cpdu) = h(cpdu) eq c_initialize_ri;
        is_c_initialize_rc(cpdu) = h(cpdu) eq c_initialize_rc;
        is_c_begin_ri(cpdu) = h(cpdu) eq c_begin_ri;
        is_c_begin_rc(cpdu) = h(cpdu) eq c_begin_rc;
        is_c_prepare_ri(cpdu) = h(cpdu) eq c_prepare_ri;
        is_c_ready_ri(cpdu) = h(cpdu) eq c_ready_ri;
        is_c_commit_ri(cpdu) = h(cpdu) eq c_commit_ri;
        is_c_commit_rc(cpdu) = h(cpdu) eq c_commit_rc;
        is_c_rollback_ri(cpdu) = h(cpdu) eq c_rollback_ri;
        is_c_rollback_rc(cpdu) = h(cpdu) eq c_rollback_rc;
        is_c_recover_ri(cpdu) = h(cpdu) eq c_recover_ri;
        is_c_recover_rc(cpdu) = h(cpdu) eq c_recover_rc;
ofsort  ccr_ver
```

```
        get_v(c_initialize_ri(v)) = v;
        get_v(c_initialize_rc(v)) = v;
ofsort  atomic_action_identifier
        get_aai(c_begin_ri(aai,bs,ud)) = aai;
        get_aai(c_recover_ri(aai,bi,rs,ud)) = aai;
        get_aai(c_recover_rc(aai,bi,rr,ud)) = aai;
ofsort  branch_suffix
        get_bs(c_begin_ri(aai,bs,ud)) = bs;
ofsort  branch_identifier
        get_bi(c_recover_ri(aai,bi,rs,ud)) = bi;
        get_bi(c_recover_rc(aai,bi,rr,ud)) = bi;
ofsort  user_data
        get_ud(c_begin_ri(aai,bs,ud)) = ud;
        get_ud(c_begin_rc(ud)) = ud;
        get_ud(c_prepare_ri(ud)) = ud;
        get_ud(c_ready_ri(ud)) = ud;
        get_ud(c_commit_ri(ud)) = ud;
        get_ud(c_commit_rc(ud)) = ud;
        get_ud(c_rollback_ri(ud)) = ud;
        get_ud(c_rollback_rc(ud)) = ud;
        get_ud(c_recover_ri(aai,bi,rs,ud)) = ud;
        get_ud(c_recover_rc(aai,bi,rr,ud)) = ud;
ofsort  requestor_recovery_state
        get_rs(c_recover_ri(aai,bi,rs,ud)) = rs;
ofsort  responder_recovery_state
        get_rr(c_recover_rc(aai,bi,rr,ud)) = rr;
endtype (* CCR_PDU *)
```

10.2 Application Protocol Data Unit

This type represents the general Application Layer PDUs. Their structure are not important in this description. Therefore, a just place holder is defined.

```
type    Sub_A_PDU is General renamedby
sortnames
        APDU for general
opnnames
        a_empty for null
endtype (* Sub_A_PDU *)

type    A_PDU is Sub_A_PDU,Key,CCR_PDU
opns
        h : APDU -> key
        apdu : -> key
eqns
forall  a:APDU
ofsort  key
        h(a) = apdu;
        apdu = succ(c_recover_rc);
endtype (* A_PDU *)
```

10.3 Transformation from CCR Service Primitive to CCR PDU

The aim of this type is to define two transformation functions; `begin_trans_sp_to_pdu` and `recover_trans_sp_to_pdu`.

`begin_trans_sp_to_pdu` transforms as follows;

C-Begin request	→	`c_begin_ri`
C-Commit request + C-Begin request	→	`c_begin_ri`
C-Rollback request + C-Begin request	→	`c_begin_ri`

`recover_trans_sp_to_pdu` transforms as follows;

C-Recover request	→	`c_recover_ri`
C-Recover response	→	`c_recover_rc`

The transformation of C-Commit request and C-Rollback request service primitives and their concatenation with `c_begin_ri` are described in the process part.

The important point is the transformations of `sender/receiver` in CCR version 2. The use of `sender/receiver` is a sender's option. This is represented by the non-deterministic expressions whose premisses are not exclusive.

```
type    CCR_Service_Primitive_To_CCR_PDU is CCR_PDU,CCR_Service_Primitive,
                                            AE_Title,
                                            Transformation_User_Data
opns
        begin_trans_sp_to_pdu : CSP,ccr_ver,ae_title,ae_title -> CPDU
        recover_trans_sp_to_pdu : CSP,ccr_ver,ae_title,ae_title -> CPDU
eqns
forall  csp:CSP,ver:ccr_ver,aet1,aet2:ae_title
ofsort  CPDU
        is_CBeginReq(csp) or is_CCommitReq_CBeginReq(csp)
                          or is_CRollbackReq_CBeginReq(csp) =>
                begin_trans_sp_to_pdu(csp,ver,aet1,aet2)
                        = c_begin_ri(
                                atomic_action_identifier(
                                        masters_name(get_aaimn(csp)),
                                        get_aais(csp)),
                                get_bis(csp),
                                to_pdu(get_ud(csp)));

        (is_CBeginReq(csp) or is_CCommitReq_CBeginReq(csp)
                                or is_CRollbackReq_CBeginReq(csp))
                        and (get_aaimn(csp) eq aet1) =>
                begin_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_begin_ri(
                                atomic_action_identifier(
                                        masters_name(sender),
                                        get_aais(csp)),
                                get_bis(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverReq(csp) =>
```

```
        recover_trans_sp_to_pdu(csp,ver,aet1,aet2)
                = c_recover_ri(
                        atomic_action_identifier(
                                masters_name(get_aaimn(csp)),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(get_bisn(csp)),
                                get_bis(csp)),
                        get_rs(csp),
                        to_pdu(get_ud(csp)));

is_CRecoverReq(csp) and (get_aaimn(csp) eq aet1)
                        and (get_bisn(csp) eq aet1) =>
        recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                = c_recover_ri(
                        atomic_action_identifier(
                                masters_name(sender),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(sender),
                                get_bis(csp)),
                        get_rs(csp),
                        to_pdu(get_ud(csp)));

is_CRecoverReq(csp) and (get_aaimn(csp) eq aet2)
                        and (get_bisn(csp) eq aet2) =>
        recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                = c_recover_ri(
                        atomic_action_identifier(
                                masters_name(receiver),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(receiver),
                                get_bis(csp)),
                        get_rs(csp),
                        to_pdu(get_ud(csp)));

is_CRecoverRsp(csp) =>
        recover_trans_sp_to_pdu(csp,ver,aet1,aet2)
                = c_recover_rc(
                        atomic_action_identifier(
                                masters_name(get_aaimn(csp)),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(get_bisn(csp)),
                                get_bis(csp)),
                        get_rr(csp),
                        to_pdu(get_ud(csp)));

is_CRecoverRsp(csp) and (get_aaimn(csp) eq aet1)
                        and (get_bisn(csp) eq aet1) =>
        recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                = c_recover_rc(
                        atomic_action_identifier(
```

```
                                        masters_name(sender),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(sender),
                                        get_bis(csp)),
                                get_rr(csp),
                                to_pdu(get_ud(csp)));

        is_CRecoverRsp(csp) and (get_aaimn(csp) eq aet2)
                                        and (get_bisn(csp) eq aet2) =>
                recover_trans_sp_to_pdu(csp,ver2,aet1,aet2)
                        = c_recover_rc(
                                atomic_action_identifier(
                                        masters_name(receiver),
                                        get_aais(csp)),
                                branch_identifier(
                                        superiors_name(receiver),
                                        get_bis(csp)),
                                get_rr(csp),
                                to_pdu(get_ud(csp)));
endtype (* CCR_Service_Primitive_To_CCR_PDU *)
```

10.4 Transformation from CCR PDU to CCR Service Primitive

The aim of this type is similar to that of CCR_Service_Primitive_To_CCR_PDU. CCR_Service_Primitive_To_CCR_PDU includes the functions which transform CCR service primitives into CCR PDUs. In CCR_PDU_To_CCR_Service_Primitives, the functions which transform CCR PDUs into CCR service primitives are defined. The important point of this type is the same as that of the previous type, that is, the treatment of sender/receiver. By the facility of get_aet this is treated.

```
type    CCR_PDU_To_CCR_Service_Primitive is CCR_PDU,CCR_Service_Primitive,
                                                AE_Title,
                                                Transformation_User_Data
opns
        begin_trans_pdu_to_sp : CPDU,ccr_ver,ae_title,ae_title -> CSP
        recover_trans_pdu_to_sp : CPDU,ccr_ver,ae_title,ae_title -> CSP
        get_aaimn,
        get_bisn : CPDU,ccr_ver,ae_title,ae_title -> ae_title
        get_aais : CPDU -> atomic_action_suffix
        get_bis :  CPDU -> branch_suffix
eqns
forall  pdu:CPDU,ver:ccr_ver,aet1,aet2:ae_title
ofsort  ae_title
is_c_begin_ri(pdu) or is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_aaimn(pdu,ver,aet1,aet2) =
                get_aet(get_mn(get_aai(pdu)),ver,aet1,aet2);
is_c_begin_ri(pdu) =>
        get_bisn(pdu,ver,aet1,aet2) = aet2;      (* sender *)
is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_bisn(pdu,ver,aet1,aet2) =
                get_aet(get_sn(get_bi(pdu)),ver,aet1,aet2);
ofsort  atomic_action_suffix
is_c_begin_ri(pdu) or is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
```

```
        get_aais(pdu)    = get_aas(get_aai(pdu));
ofsort  branch_suffix
is_c_begin_ri(pdu) =>
        get_bis(pdu)     = get_bs(pdu);
is_c_recover_ri(pdu) or is_c_recover_rc(pdu) =>
        get_bis(pdu)     = get_bs(get_bi(pdu));
ofsort  CSP
        begin_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                = CBeginInd(get_aaimn(pdu,ver,aet1,aet2),
                            get_aais(pdu),
                            get_bisn(pdu,ver,aet1,aet2),
                            get_bis(pdu),
                            to_sp(get_ud(pdu)));

        is_c_recover_ri(pdu) =>
                recover_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                        = CRecoverInd(get_rs(pdu),
                                      get_aaimn(pdu,ver,aet1,aet2),
                                      get_aais(pdu),
                                      get_bisn(pdu,ver,aet1,aet2),
                                      get_bis(pdu),
                                      to_sp(get_ud(pdu)));

        is_c_recover_rc(pdu) =>
                recover_trans_pdu_to_sp(pdu,ver,aet1,aet2)
                        = CRecoverCnf(get_rr(pdu),
                                      get_aaimn(pdu,ver,aet1,aet2),
                                      get_aais(pdu),
                                      get_bisn(pdu,ver,aet1,aet2),
                                      get_bis(pdu),
                                      to_sp(get_ud(pdu)));
endtype (* CCR_PDU_To_CCR_Service_Primitive *)
```

10.5 Key Queue Type

The element of a sort key_queue is a list of keys. The operators _._ and _+_ on key_queue are used for the operations of concatenation or deconcatenation of PDUs in between CCR and the presentation layer. In ConcDeconcType this operation plays an important role when the operation may_conc is defined.

```
type    Key_Queue is Key
sorts   key_queue
opns
        kq_empty : -> key_queue
        _._ : key,key_queue -> key_queue
        _+_ : key_queue,key -> key_queue
eqns
forall  k,k1:key,kq:key_queue
ofsort  key_queue
        kq_empty + k = k . kq_empty;
        (k . kq) + k1 = k . (kq + k1);
endtype (* Key_Queue *)
```

10.6 PDU Type

This is a general PDU type in this description. The structure of a general PDU is similar to a general service primitive. For the use of key_queue in the definition of may_conc in ConcDeconcType, the mapping function h into key_queue is defined. The key is an abstraction of each general PDU. The argument of general PDU, a real PDU, may have some parameters, but mapped result by h, key, has no parameter.

```
type    Protocol_Data_Unit is CCR_PDU,A_PDU
sorts   PDU
opns
        empty : -> PDU
        pdu : CPDU -> PDU
        pdu : APDU -> PDU
        is_empty : PDU -> Bool
        is_ccr_pdu : PDU -> Bool
        is_ase_pdu : PDU -> Bool
        get_cpdu  : PDU -> CPDU
        get_apdu  : PDU -> APDU
        h : PDU -> key
eqns
forall  c:CPDU,a:APDU
ofsort  Bool
        is_empty(empty) = true;
        is_empty(pdu(c)) = false;
        is_empty(pdu(a)) = false;
        is_ccr_pdu(empty) = false;
        is_ccr_pdu(pdu(c)) = true;
        is_ccr_pdu(pdu(a)) = false;
        is_ase_pdu(empty) = false;
        is_ase_pdu(pdu(c)) = false;
        is_ase_pdu(pdu(a)) = true;
ofsort  CPDU
        get_cpdu(pdu(c)) = c;
ofsort  APDU
        get_apdu(pdu(a)) = a;
ofsort  key
        h(pdu(c)) = h(c);
        h(pdu(a)) = h(a);
endtype (* Protocol_Data_Unit *)
```

10.7 PDU Queue Type

In this type, the quueue of PDUs and the operations for the queue are defined. The operation h is used for the mapping of PDU queue to key quueue in the definition may_conc in ConcDeconcType.

```
type    PDU_Queue is Protocol_Data_Unit,Key_Queue
sorts   PDUqueue
opns
        q_empty : -> PDUqueue
        _._ : PDU,PDUqueue -> PDUqueue
        head : PDUqueue -> PDU
        tail : PDUqueue -> PDUqueue
```

```
        queue : PDU -> PDUqueue
        _+_ : PDUqueue,PDU -> PDUqueue
        is_q_empty,is_PDU : PDUqueue -> Bool
        h : PDUqueue -> key_queue
eqns
forall  p,p1:PDU,q:PDUqueue
ofsort  PDU
        head(p.q) = p;
ofsort  PDUqueue
        tail(p.q) = q;
        queue(empty) = q_empty;
        queue(p) = p.q_empty;
        q_empty + p = queue(p);
        (p.q) + p1 = p.(q + p1);
ofsort  bool
        is_q_empty(q_empty) = true;
        is_q_empty(p.q) = false;
        is_PDU(q) = not(is_q_empty(q)) and is_q_empty(tail(q));
ofsort  key_queue
        h(p.q) = h(p).h(q);
        h(q_empty) = kq_empty;
endtype (* PDU_Queue *)
```

10.8 Search CCR PDU Type

In the process part of this description, the operation which searches a CCR PDU and which obtains it is necessary. This type gives the operation.

```
type    Search_CCR_PDU is PDU_Queue
opns
        ccr_pdu : PDUqueue -> CPDU
        ase_pdu : PDUqueue -> PDUqueue
eqns
forall  p:PDU,pq:PDUqueue
ofsort  CPDU
        ccr_pdu(q_empty) = c_empty;
        is_ccr_pdu(p) =>
                ccr_pdu(p.pq) = get_cpdu(p);
        not(is_ccr_pdu(p)) =>
                ccr_pdu(p.pq) = ccr_pdu(pq);
ofsort  PDUqueue
        ase_pdu(q_empty) = q_empty;
        is_ccr_pdu(p) =>        ase_pdu(p.pq) = ase_pdu(pq);
        not(is_ccr_pdu(p)) =>   ase_pdu(p.pq) = p.ase_pdu(pq);
endtype (* Search_CCR_PDU *)
```

10.9 Concatination and Deconcatination Type

The operator may_conc is a predicate: it is true if the PDUs may be concatenated. This rule should be the same as the description of ISO/IEC 9805-1, clause 10.

```
type    ConcDeconcType is Protocol_Data_Unit,PDU_Queue
```

```
opns
        may_conc : key_queue,key -> Bool
        is_empty_seq, is_apdu_seq,
        is_c_begin_ri_seq, is_c_begin_rc_seq,
        is_c_prepare_ri_seq, is_c_ready_ri_seq,
        is_c_commit_ri_seq, is_c_commit_rc_seq, is_c_commit_rc_seq_tail,
        is_c_rollback_ri_seq, is_c_rollback_rc_seq,
        is_c_begin_ri_pdu, is_c_begin_rc_pdu : key_queue -> Bool
eqns
forall  k:key,kq:key_queue
ofsort  Bool
        may_conc(kq,k)  =  is_empty_seq(kq)
                        or is_apdu_seq(kq + k)
                        or is_c_begin_ri_seq(kq + k)
                        or is_c_begin_rc_seq(kq + k)
                        or is_c_prepare_ri_seq(kq + k)
                        or is_c_ready_ri_seq(kq + k)
                        or is_c_commit_ri_seq(kq + k)
                        or is_c_commit_rc_seq(kq + k)
                        or is_c_rollback_ri_seq(kq + k)
                        or is_c_rollback_rc_seq(kq + k);

        is_empty_seq(kq_empty)  = true;
        is_empty_seq(k.kq)      = false;

        is_apdu_seq(kq_empty)   = true;
        is_apdu_seq(k.kq)       = (k eq apdu) and is_apdu_seq(kq);

        is_c_begin_ri_seq(kq_empty)     = false;
        is_c_begin_ri_seq(k.kq)         =
                ((k eq apdu) and is_c_begin_ri_seq(kq)) or
                ((k eq c_begin_ri) and
                        (is_c_prepare_ri_seq(kq) or is_apdu_seq(kq)));

        is_c_begin_rc_seq(kq_empty)     = false;
        is_c_begin_rc_seq(k.kq)         =
                ((k eq apdu) and is_c_begin_rc_seq(kq)) or
                ((k eq c_begin_rc) and
                        (is_c_ready_ri_seq(kq) or
                                (is_c_ready_ri_seq(kq) or is_apdu_seq(kq))));

        is_c_prepare_ri_seq(kq_empty)   = false;
        is_c_prepare_ri_seq(k.kq)       =
                ((k eq apdu) and is_c_prepare_ri_seq(kq)) or
                ((k eq c_prepare_ri) and is_empty_seq(kq));

        is_c_ready_ri_seq(kq_empty)     = false;
        is_c_ready_ri_seq(k.kq)         =
                ((k eq apdu) and is_c_ready_ri_seq(kq)) or
                ((k eq c_ready_ri) and is_empty_seq(kq));

        is_c_commit_ri_seq(kq_empty)    = false;
        is_c_commit_ri_seq(k.kq)        =
                (k eq c_commit_ri) and is_c_begin_ri_pdu(kq);
```

```
        is_c_commit_rc_seq(k.kq)        =
                (k eq c_commit_rc) and is_c_commit_rc_seq_tail(kq);

        is_c_commit_rc_seq_tail(kq_empty)       = false;
        is_c_commit_rc_seq_tail(k.kq)           =
                ((k eq c_begin_rc) and
                        (is_c_ready_ri_seq(kq) or is_apdu_seq(kq))) or
                is_c_ready_ri_seq(k.kq) or is_apdu_seq(k.kq);

        is_c_rollback_ri_seq(kq_empty)  = false;
        is_c_rollback_ri_seq(k.kq)      =
                (k eq c_rollback_ri) and is_c_begin_ri_pdu(kq);

        is_c_rollback_rc_seq(kq_empty)  = false;
        is_c_rollback_rc_seq(k.kq)      =
                (k eq c_rollback_rc) and
                        (is_c_begin_ri_pdu(kq) or is_c_begin_rc_pdu(kq) or
                         is_apdu_seq(kq));

        is_c_begin_ri_pdu(kq_empty)     = false;
        is_c_begin_ri_pdu(k.kq)         =
                (k eq c_begin_ri) and is_empty_seq(kq);

        is_c_begin_rc_pdu(kq_empty)     = false;
        is_c_begin_rc_pdu(k.kq)         =
                (k eq c_begin_rc) and is_empty_seq(kq);
endtype (* ConcDeconcType *)
```

10.10 CCR Version Number Types

This type includes the definition of CCR version numbers. Four elements are defined as follows;

```
        ver1      :  the protocol machine supports only version 1
        ver2      :  the protocol machine supports only version 2
        ver12     :  the protocol machine supports both version 1 and version 2
        rejected  :  it represents the failure of version negatiation between the peer AEIs
```

The operator `select_ver` returns the negotiated version number.

```
type    CCR_Version_Number is Boolean,Key
sorts   ccr_ver
opns
        ver1,ver2,ver12,rejected : -> ccr_ver
        select_ver : ccr_ver,ccr_ver -> ccr_ver
        _contains_ : ccr_ver,ccr_ver -> bool
        _eq_,_ne_ : ccr_ver,ccr_ver -> bool
        h : ccr_ver -> key
eqns
forall  ver,vers:ccr_ver
ofsort  key
        h(rejected) = 0;
```

```
        h(ver1) = succ(h(rejected));
        h(ver2) = succ(h(ver1));
        h(ver12) = succ(h(ver2));
ofsort  Bool
        ver eq vers = h(ver) eq h(vers);
        ver ne vers = not(ver eq vers);
        ver12 contains ver = ver ne rejected;
        vers ne ver12 =>
                vers contains ver = vers eq ver;
ofsort  ccr_ver
        (ver contains ver2) and (vers contains ver2) =>
                select_ver(ver,vers) = ver2;
        not((ver contains ver2) and (vers contains ver2))
        and (ver contains ver1) and (vers contains ver1) =>
                select_ver(ver,vers) = ver1;
        not((ver contains ver2) and (vers contains ver2))
        and not((ver contains ver1) and (vers contains ver1)) =>
                select_ver(ver,vers) = rejected;
endtype (* CCR_Version_Number *)
```

10.11 Atomic Action Branch Type

This type represents Atomic Action Branch. Atomic Action Branch may be omitted, so aab_null is also defined.

```
type    Atomic_Action_Branch is Atomic_Action_Identifier,Branch_Identifier,
                                        CCR_PDU_To_CCR_Service_Primitive
sorts   atomic_action_branch
opns
        aab_null : -> atomic_action_branch
        atomic_action_branch : atomic_action_identifier,
                               branch_identifier -> atomic_action_branch
        get_aai : atomic_action_branch -> atomic_action_identifier
        get_bi : atomic_action_branch -> branch_identifier
        _eq_,_ne_ : atomic_action_branch,atomic_action_branch -> Bool
        get_aab :  CPDU,ccr_ver,ae_title,ae_title -> atomic_action_branch
        get_aab :  CSP  -> atomic_action_branch
eqns
forall  aab,aab1:atomic_action_branch,
        pdu:CPDU,ver:ccr_ver,aet1,aet2:ae_title,csp:CSP,
        aai,aai1:atomic_action_identifier,bi,bi1:branch_identifier
ofsort  atomic_action_identifier
        get_aai(atomic_action_branch(aai,bi)) = aai;
ofsort  branch_identifier
        get_bi(atomic_action_branch(aai,bi)) = bi;
ofsort  Bool
        aab_null eq aab_null = true;
        aab_null eq atomic_action_branch(aai,bi) = false;
        atomic_action_branch(aai,bi) eq aab_null = false;
        atomic_action_branch(aai,bi) eq atomic_action_branch(aai1,bi1)
                = (aai eq aai1) and (bi eq bi1);
        aab ne aab1 = not(aab eq aab1);
ofsort  atomic_action_branch
```

```
        get_aab(pdu,ver,aet1,aet2) =
                atomic_action_branch(
                        atomic_action_identifier(
                                masters_name(get_aaimn(pdu,ver,aet1,aet2)),
                                get_aais(pdu)),
                        branch_identifier(
                                superiors_name(get_bisn(pdu,ver,aet1,aet2)),
                                get_bis(pdu)));
        get_aab(csp) =
                atomic_action_branch(
                        atomic_action_identifier(
                                masters_name(get_aaimn(csp)),
                                get_aais(csp)),
                        branch_identifier(
                                superiors_name(get_bisn(csp)),
                                get_bis(csp)));
endtype (* Atomic_Action_Branch *)
```

10.12 Failure Event Type

Failure represents a failure event. This failure is general. In CCR protocol specification, the kind of failures is not significant.

```
type    Failure is
sorts   failure
opns
        failure : -> failure
endtype (* Failure *)
```

11 Global constraints of the CCR protocol specification behaviour

Two gates ccr_u and ccr_l are used to communicate in between two processes CF (Control Function) and CCR (CCR protocol machine).

```
behaviour
        hide ccr_u,ccr_l in
        ( CCR[ccr_u,ccr_l,env](ver)
          |[ccr_u,ccr_l,env]|
          CF[u,p,ccr_u,ccr_l,acse_u,acse_l,env,u_env]
        )
where
```

12 CCR Protocol Machine

Two cases are described in process CCR. The first is the case that the negotiation of CCR protocol version is necessary. The process CCRVersionNegotiate works for this purpose. The second case is that the negotiation of version numbers finished in advance and that the environment (e.g. CCR user) knows

the negotiated version. This case may occur after some failures happened. It is treated by the process CCRFixedVersion.

```
process CCR[c_u,c_l,env](ver:ccr_ver) : exit :=
        (         CCRVersionNegotiate[c_u,c_l](ver)
          []
          ( env?fixed_ver:ccr_ver?aet_self:AE_title?aet_peer:AE_title;
            CCRFixedVersion[c_u,c_l](fixed_ver,aet_self,aet_peer)
        ) ) [> env!failure;exit
where
```

12.1 CCR Protocol Version Negotiation

The negotiation of CCR protocol version is necessary when the protocol machine has functionalities of version 2. If it has only functionalities of version 1, the negotiation is meaingless. In the process CCRInitialize the negotiation of version is carried out. After that CCRPM can start.

```
process CCRVersionNegotiate[c_u,c_l](ver:ccr_ver) : exit :=
        c_u!ver;
        ( ( [ver eq ver1]->exit(ver1)
            []
            [ver contains ver2]->CCRInitialize[c_u,c_l](ver)
          ) >> accept fixed_ver:ccr_ver in
                ( [fixed_ver eq rejected]->exit
                  []
                  [fixed_ver ne rejected]->
                                c_u?aet_self:AE_title?aet_peer:AE_title;
                                CCRPM[c_u,c_l](fixed_ver,aet_self,aet_peer)
        )       )
where
```

12.2 Process for CCR Initialize

CCRIitialize negotiates CCR protocol version. Two PDUs, c_initiaize_ri and c_initialize_rc, concern the negotiation. If the peer entity has only version 1, these PDUs are not accepted and the response PDU is empty.

```
process CCRInitialize[c_u,c_l](ver:ccr_ver) : exit(ccr_ver) :=
        ( c_u!AAssoc_req;
          c_l!c_initialize_ri(ver);
          ( ( c_l?pdu:CPDU[is_c_initialize_rc(pdu)];
              c_u!select_ver(ver,get_v(pdu));
              exit(select_ver(ver,get_v(pdu)))
            )
            []
            ( c_l!c_empty;
              c_u!select_ver(ver,ver1);
              exit(select_ver(ver,ver1))
        ) ) )
        []
        ( ( ( c_l?pdu:CPDU[is_c_initialize_ri(pdu)];
```

```
                c_u!select_ver(ver,get_v(pdu));
                stop
              )
             []
             ( c_l!c_empty;
               c_u!select_ver(ver,ver1);
               stop
          ) )
          [> ( c_u?fixed_ver:ccr_ver;
               ( [fixed_ver eq rejected]->exit(fixed_ver)
                 []
                 [fixed_ver ne rejected]->
                 ( c_u!AAssoc_rsp;
                   c_l!c_initialize_rc(fixed_ver);
                   exit(fixed_ver)
        ) ) ) )
endproc (* CCRInitialize *)

endproc (* CCRVersionNegotiate *)
```

12.3 CCR Fixed Version Process

When the negotiation of CCR protocol version finished in advance and the environment (e.g. CCR user) knows an available version, this process works. This case happens, e.g., in the recovery case.

```
process CCRFixedVersion[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title)
                                                                    : noexit :=
        CCRPM[c_u,c_l](ver,aet_self,aet_peer)
endproc (* CCRFixedVersion *)
```

12.4 Process for CCR Behaviour

This process describes CCR the behaviour of a protocol machine.

```
process CCRPM[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title) : noexit :=
        (    SupCCR[c_u,c_l](ver,aet_self,aet_peer)
          [] SubCCR[c_u,c_l](ver,aet_self,aet_peer)
        ) >> CCRPM[c_u,c_l](ver,aet_self,aet_peer)
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR End Point :                   *)
(*-------------------------------------------------------------*)
process SupCCR[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title) : exit :=
        SupAction[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
        []
        SupRecovery[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCRAction : Normal                *)
```

```
(*      C-BEGIN req                                                    *)
(*-------------------------------------------------------------------*)
process SupAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                           c_br,n_br:atomic_action_branch) : exit :=
        c_u?sp:CSP[is_CBeginReq(sp)];
        c_l!begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
        ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),n_br)
where

(*-------------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal                     *)
(*      C-BEGIN cnf ..... C-PREPARE req                              *)
(*-------------------------------------------------------------------*)
process ContSupAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                           c_br,n_br:atomic_action_branch): exit :=
        ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
          c_u!CBeginCnf(to_sp(get_ud(pdu)));
          ( ( c_u?sp:CSP[is_CPrepareReq(sp)];
              c_l!c_prepare_ri(to_pdu(get_ud(sp)));
              ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ( c_u?sp:CSP[is_CPrepareReq(sp)];
          c_l!c_prepare_ri(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
              c_u!CBeginCnf(to_sp(get_ud(pdu)));
              ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
where

(*-------------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal                     *)
(*      C-READY req or C-ROLLBACK                                    *)
(*-------------------------------------------------------------------*)
process ContSupAction1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                           c_br,n_br:atomic_action_branch): exit :=
        ( c_l?pdu:CPDU[is_c_ready_ri(pdu)];
          c_u!CReadyInd(to_sp(get_ud(pdu)));
          SupDecision[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
```

```
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          SupRollSend1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        (*      When C-Rollback request + C-Begin request is recived,   *)
        (*      CCRPM simultaneously issues c-rollback-ri and           *)
        (*      c-begin-ri at c_l gate.                                 *)
        ( c_u?sp:CSP[is_CRollbackReq_CBeginReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SupRollsend2[c_u,c_l](ver,aet_self,aet_peer,c_br,get_aab(sp))
        )
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal               *)
(*      C-COMMIT req or C-ROLLBACK req                         *)
(*-------------------------------------------------------------*)
process SupDecision[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                             c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CCommitReq(sp)];
          c_l!c_commit_ri(to_pdu(get_ud(sp)));
          c_l?pdu:CPDU[is_c_commit_rc(pdu)];
          c_u!CCommitCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        (*      When C-Commit request + C-Begin request is recived,     *)
        (*      CCRPM simultaneously issues c-commit-ri and c-begin-ri  *)
        (*      at c_l gate.                                            *)
        ( c_u?sp:CSP[is_CCommitReq_CBeginReq(sp)];
          c_l!c_commit_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          c_l?pdu:CPDU[is_c_commit_rc(pdu)];
          c_u!CCommitCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
        []
        (*      When C-Rollback request + C-Begin request is recived,   *)
        (*      CCRPM simultaneously issues c-rollback-ri and           *)
        (*      c-begin-ri at c_l gate.                                 *)
        ( c_u?sp:CSP[is_CRollbackReq_CBeginReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)))
             !begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
```

```
          c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
endproc (* SupDecision *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal               *)
(*      C-ROLLBACK-RC or C-ROLLBACK collision                  *)
(*                              After C-ROLLBACK req           *)
(*-------------------------------------------------------------*)
process SupRollSend1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                              c_br,n_br:Atomic_Action_Branch) : exit :=
        [ver eq ver2]->
        ( c_l?pdu:CPDU[is_c_begin_rc(pdu)];
          SupRollSend1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
endproc (* SupRollSend1 *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Action : Normal               *)
(*      C-ROLLBACK-RC or C-ROLLBACK collision                  *)
(*                      After C-ROLLBACK req + C-BEGIN req     *)
(*-------------------------------------------------------------*)
process SupRollSend2[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                              c_br,n_br:Atomic_Action_Branch) : exit :=
        ( c_l?pdu:CPDU[is_c_rollback_rc(pdu)];
          c_u!CRollbackCnf(to_sp(get_ud(pdu)));
          ContSupAction[c_u,c_l](ver,aet_self,aet_peer,n_br,aab_null)
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          c_l!begin_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          ContSupAction1[c_u,c_l](ver,aet_self,aet_peer,n_br,aab_null)
        )
endproc (* SupRollSend2 *)

endproc (* ContSupAction1 *)
```

```
endproc (* ContSupAction *)

endproc (* SupAction *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER req or C-RECOVER-RI                          *)
(*-------------------------------------------------------------*)
process SupRecovery[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title,
                             c_br,n_br:Atomic_Action_Branch) : exit :=
        ( c_l?pdu:CPDU[is_c_recover_ri(pdu) and (get_rs(pdu) eq ready)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SupRecResponse[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),aab_null)
        )
        []
        ( c_u?sp:CSP[is_CRecoverReq(sp) and (get_rs(sp) eq commit)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          SupRecActions[c_u,c_l](ver,aet_self,aet_peer,get_aab(sp),aab_null)
        )
where

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER(ready)-RI APDU received                      *)
(*-------------------------------------------------------------*)
process SupRecResponse[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                             c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CRecoverReq(sp) and (get_rs(sp) eq commit)];
          ( [c_br eq get_aab(sp) ] ->
            ( c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
              SupRecActions[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) ) )
        []
        ( c_u?sp:CSP[is_CRecoverRsp(sp) and (get_rr(sp) eq retry)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          i;      (* i works as a retry timer *)
          SupRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ( c_u?sp:CSP[is_CRecoverRsp(sp) and (get_rr(sp) eq unknown)];
          c_l!recover_trans_sp_to_pdu(sp,ver,aet_self,aet_peer);
          exit
        )
endproc (* SupRecResponse *)

(*-------------------------------------------------------------*)
(*      Process for Superior CCR Recovery : Recover            *)
(*      C-RECOVER-RC                                           *)
(*-------------------------------------------------------------*)
process SupRecActions[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                             c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq done)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
```

```
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_recover_rc(pdu) and (get_rr(pdu) eq retry)];
          c_u!recover_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          SupRecovery[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
endproc (* SupRecActions *)

endproc (* SupRecovery *)

endproc (* SupCCR *)

(*-------------------------------------------------------------------*)
(*      Process for Subordinate CCR End Point                        *)
(*-------------------------------------------------------------------*)
process SubCCR[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:AE_title) : exit :=
        SubAction[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
        []
        SubRecovery[c_u,c_l](ver,aet_self,aet_peer,aab_null,aab_null)
where

(*-------------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal                  *)
(*      C-BEGIN-RI                                                   *)
(*-------------------------------------------------------------------*)
process SubAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                           c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_begin_ri(pdu)];
          c_u!begin_trans_pdu_to_sp(pdu,ver,aet_self,aet_peer);
          ContSubAction[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),n_br)
        )
where

(*-------------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal                  *)
(*      C-BEGIN rsp ... C-PREPARE-RI                                 *)
(*-------------------------------------------------------------------*)
process ContSubAction[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                           c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CBeginRsp(sp)];
          c_l!c_begin_rc(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
              c_u!CPrepareInd(to_sp(get_ud(pdu)));
              ContSubAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ( c_u?sp:CSP[is_CReadyReq(sp)];
              c_l!c_ready_ri(to_pdu(get_ud(sp)));
              ContSubAction2[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
```

```
        ) )
        []
        ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
          c_u!CPrepareInd(to_sp(get_ud(pdu)));
          ( ( c_u?sp:CSP[is_CBeginRsp(sp)];
              c_l!c_begin_rc(to_pdu(get_ud(sp)));
              ContSubAction1[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            ( c_u?sp:CSP[is_CReadyReq(sp)];
              c_l!c_ready_ri(get_ud(pdu));
              ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
            )
            []
            SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        ) )
        []
        ( c_u?sp:CSP[is_CReadyReq(sp)];
          c_l!c_ready_ri(to_pdu(get_ud(sp)));
          ContSubAction2[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
where

(*-------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal            *)
(*      After C-BEGIN rsp and C-PREPARE ind                    *)
(*-------------------------------------------------------------*)
process ContSubAction1[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_u?sp:CSP[is_CReadyReq(sp)];
          c_l!c_ready_ri(to_pdu(get_ud(sp)));
          ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        SubRollBack[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
endproc (* ContSubAction1 *)

(*-------------------------------------------------------------*)
(*      Process for Supbordinate CCR Action : Normal           *)
(*      After C-READY req and not C-PREPARE ind                *)
(*-------------------------------------------------------------*)
process ContSubAction2[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_prepare_ri(pdu)];
          c_u!CPrepareInd(to_sp(get_ud(pdu)));
          ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
        )
        []
        ContSubAction3[c_u,c_l](ver,aet_self,aet_peer,c_br,n_br)
endproc (* ContSubAction2 *)

(*-------------------------------------------------------------*)
```

```
(*      Process for Subordinate CCR Action : Normal                *)
(*      After C-READY req                                          *)
(*-----------------------------------------------------------------*)
process ContSubAction3[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_commit_ri(pdu)];
          c_u!CCommitInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CCommitRsp(sp)];
          c_l!c_commit_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_l?pdu,pdu1:CPDU[is_c_commit_ri(pdu) and is_c_begin_ri(pdu1)];
          c_u!CCommitInd(to_sp(get_ud(pdu)));
          c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
          c_u?sp:CSP[is_CCommitRsp(sp)];
          c_l!c_commit_rc(to_pdu(get_ud(sp)));
          ContSubAction[c_u,c_l](ver,aet_self,aet_peer,
                        get_aab(pdu,ver,aet_self,aet_peer),aab_null)
        )
        []
        ( c_l?pdu,pdu1:CPDU[is_c_rollback_ri(pdu) and is_c_begin_ri(pdu1)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u!begin_trans_pdu_to_sp(pdu1,ver,aet_self,aet_peer);
          ContSubAction4[c_u,c_l](ver,aet_self,aet_peer,
                          c_br, get_aab(pdu1,ver,aet_self,aet_peer))
        )
endproc (* ContSubAction3 *)

(*-----------------------------------------------------------------*)
(*      Process for Subordinate CCR Action : Normal                *)
(*      C-ROLLBACK Sequence                                        *)
(*-----------------------------------------------------------------*)
process SubRollBack[c_u,c_l](ver:ccr_ver,aet_self,aet_peer:ae_title,
                                c_br,n_br:atomic_action_branch) : exit :=
        ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
          c_u!CRollbackInd(to_sp(get_ud(pdu)));
          c_u?sp:CSP[is_CRollbackRsp(sp)];
          c_l!c_rollback_rc(to_pdu(get_ud(sp)));
          exit
        )
        []
        ( c_u?sp:CSP[is_CRollbackReq(sp)];
          c_l!c_rollback_ri(to_pdu(get_ud(sp)));
          ( ( c_l?pdu:CPDU[is_c_rollback_ri(pdu)];
              c_u!CRollbackInd(to_sp(get_ud(pdu)));
              c_u?sp:CSP[is_CRollbackRsp(sp)];
```